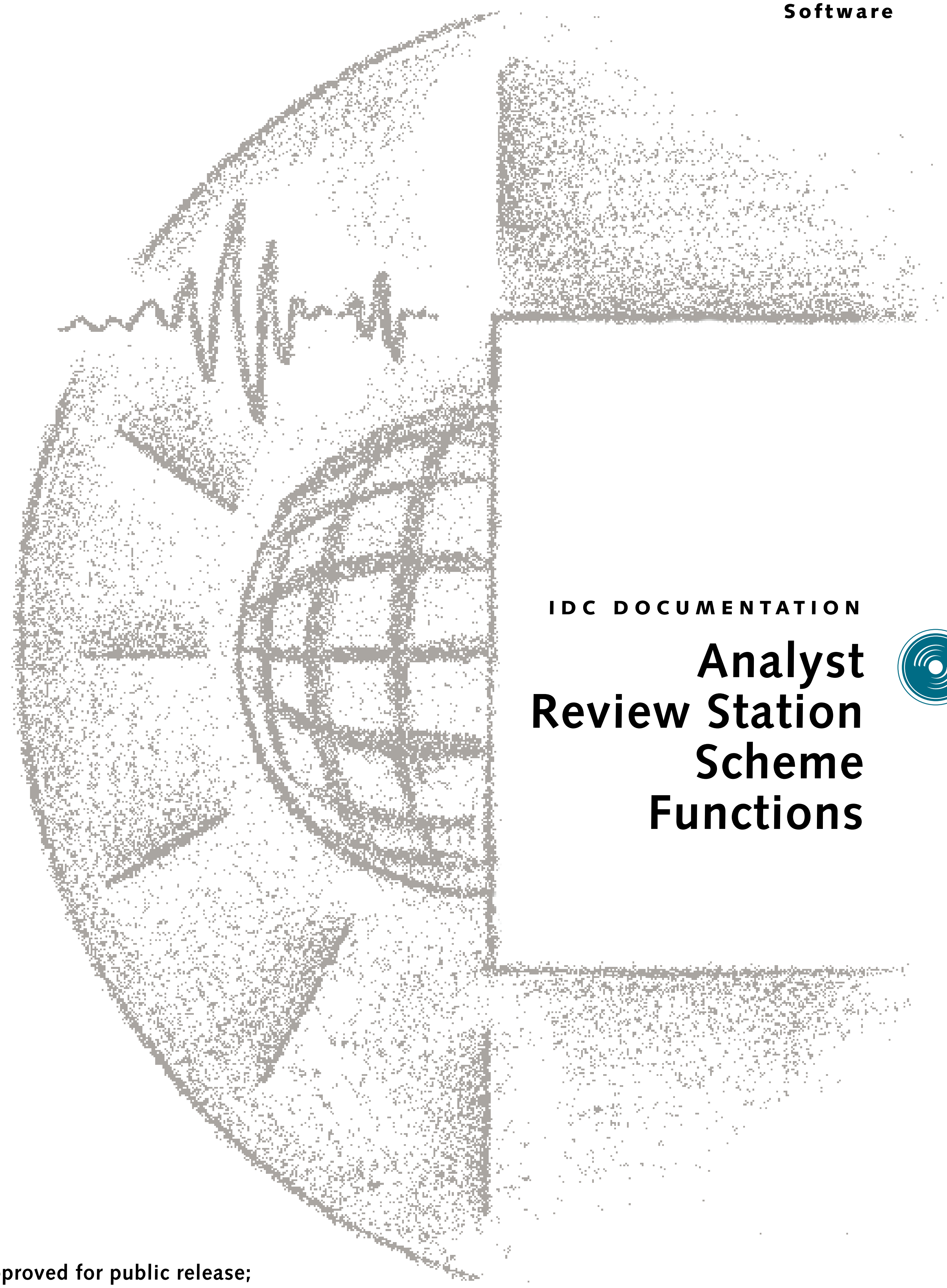

IDC DOCUMENTATION

Analyst Review Station Scheme Functions

Contributors
Greg W. Beall, Science Applications International Corporation
William E. Farrell, Science Applications International Corporation
Jeffrey Nolten, Science Applications International Corporation
Michael Skov, Science Applications International Corporation
Jim Wang, Science Applications International Corporation

Ordering Information
This document was issued by the Monitoring Systems Operation of Science Applications International Corporation (SAIC) as part of the International Data Centre (IDC) Documentation. The ordering number for this document is SAIC-99/3026, published March 1999. Copies of this document may be ordered by FAX: (619) 458-4993.

This document is cited within other IDC documents as [IDC7.2.2].

ARS Scheme Functions

CONTENTS

ARS Scheme Functions

TABLES

About this Document

This section describes the organization and content of the document and includes the following topics:

- Purpose
- Scope
- Audience
- Related Information
- Using this Document

About this Document

PURPOSE

This document describes the use of *Scheme* as the command language interface in the *Analyst Review Station* (*ARS*). The command language interface provides an interpretive layer for accessing and modifying *ARS* functionality.

SCOPE

This document defines the functions, variables, syntax, and conventions for all *ARS Scheme* commands. It describes how to use the *Scheme* interpreter in its terminal window. This document also describes the roll of the command language interface in the *ARS* system.

A general familiarity with the *ARS* application and the use of its Graphical User Interface (GUI) is assumed. Installation, configuration, and user instructions for *ARS* are provided in other documentation.

AUDIENCE

This document is intended for *ARS* configurers, developers, and skilled users.

RELATED INFORMATION

The following documents complement this manual:

- *Analyst Review Station User's Manual* [Wan96]
- *Analyst Instructions for Seismic, Hydroacoustic, and Infrasonic Data* [IDC6.2.5]

See "References" on page 229 for a list of documents that supplement this document.

USING THIS DOCUMENT

This document is located within Category 6, Technical Instructions, of the overall documentation architecture, as charted on the Roadmap located on the pages preceding the Table of Contents. This document is organized as follows:

- General Information

  This chapter describes how *Scheme* is used as the command language interface in *ARS* and provides a high-level introduction to using the *Scheme* interpreter.

- Scheme Intrinsic Functions

  This chapter describes the intrinsic *Scheme* functions including standard math functions and the general purpose functions provided in the *Scheme* library.

- Scheme Variables

  This chapter describes the *Scheme* variables including simple variables, list variables, and *C* variables.

- ARS-specific Scheme Functions

  This chapter describes the functions that are defined in the *ARS* specific *Scheme* files and compiled into the *ARS* system.

- References

  This section lists the sources cited in this document.

- Appendix: Prototype Analysis Tools

  This appendix provides initial documentation for a set of prototype analysis tools being distributed with the Prototype International Data Centre (PIDC) software.

- Glossary

  This section defines terms, abbreviations, and acronyms used in this document.

- Index

  This section lists topics and features provided in the document along with a page number for reference.

Conventions

Table I shows the typographical conventions used in this document.

TABLE I: TYPOGRAPHICAL CONVENTIONS

Element	Font/ Symbol	Example
database table	**bold**	**dataready**
database table and attribute when written in the dot notation		**prodtrack.***status*
logical hosts that appear in UNIX commands	*italics*	`% rlogin` *FSHOST* `-l cmss`
database attributes		*status*
programs, processes, software units, and libraries		*Tuxedo*
user-defined arguments and variables used in parameter (par) files or program command lines		`delete-remarks` *object*
titles of documents		*Subscription Subsystem Software User Manual*
computer code and output	`courier`	`>(list ‘a ‘b ‘c)`
filenames, directories, and web-sites		`messages@pidc.org`
text that should be typed in exactly as shown		`edit-filter-dialog`
Scheme functions		`(align-channel)`

Table II defines terms that are used in a specific context in this document. See the Glossary for a more general listing of terms, abbreviations, and acronyms.

TABLE II: TERMINOLOGY

Term	Description
argument	data serving as input to a function, or an expression which, when evaluated, will produce that data
atom	any *Scheme* object that is not a list; any datum such as a number or a string is an atom; also, the empty list, NIL
binding	attaching a value to a variable or symbol; in this document binding often refers to linking a *Scheme* function name to a compiled *C* function
element	one member of a group or set
expression	syntactically complete object such that the *Scheme* interpreter can evaluate it and return a result
lambda	marker that indicates a list is a lambda expression and is to be interpreted as a description of a function
lambda expression	list that describes a function; its first element must be the symbol LAMBDA, its second element must be an argument list, and its remaining elements constitute the body of the function
list	linked set of data elements; in *Scheme*, a list is technically a chain of cons cells where a cons cell is a unit of memory holding two pointers, one to a data element and one to the next cons cell; the data elements may be atoms or lists
NIL	symbol that represents "false" in *Scheme*; also the empty list, ( ); thus it is both a symbol and a list, and it evaluates to itself
parameter	same as argument

TABLE II: TERMINOLOGY (CONTINUED)

Term	Description
pointer	link to an object, which gives the address of that object in memory
predicate	function that answers a question by returning `t` (or some non-NIL value) for true or NIL for false
predicate expression	expression whose value is interpreted as true or false
recursion	function definition that contains a reference to itself (that is, it calls itself when executed); recursive functions are widely used in *Scheme*
set	unordered collection of elements
string	sequence of characters enclosed in double quotes
symbol	fundamental *Scheme* data type. Besides serving as data, symbols also serve as names for things such as functions or variables; as data, each symbol contains both a name cell that holds a pointer to the character string (that is, the symbol's name) and a value or function cell that holds pointers to variables or functions
variable	place where a datum is stored; variables are named by symbols

General Information

This chapter describes the use of *Scheme* in *ARS* and includes the following topics:

- Background
- Lisp, Scheme, and SIOD
- Use of Scheme in ARS
- ARS's Initialization of Scheme
- Using Scheme
- Naming Conventions
- Loading Scheme Code from Files

General Information

BACKGROUND

ARS was designed to be configurable and extensible. This capability is facilitated by a *Scheme* command language interface. The *Scheme* interface provides an interpreted software layer and an interactive text window that is accessible while *ARS* runs. While the main user interface to *ARS* is through its interactive graphical display window, and the bulk of the *ARS* software is written in the compiled *C* programming language, the interpreted *Scheme* software provides the "glue" that connects actions in the graphical interface to the compiled code.

The *Scheme* interface supports the flexibility of *ARS* in several ways:

- All significant configuration parameters are accessible and modifiable through the *Scheme* interface.
- All *ARS* functionality is accessible through *Scheme* function calls.
- *Scheme* functionality can be defined and redefined in one or more text files that can be maintained locally.
- *Scheme* definition files are loaded dynamically when *ARS* starts and additional files can be loaded at any time.
- The *Scheme* interface allows users to build functions to execute experimental, frequently used, or application-specific command sequences.
- *Scheme* definition files define all *ARS*'s interface functionality, including graphical display window menus and toolbar buttons.

The general paradigm employed in the *ARS* software architecture is to build the computationally intensive functions in *C* and provide *Scheme* entry points. High-level functions are then assembled using *Scheme*. The *Scheme* functions are activated by the menus and toolbar buttons in the graphical *ARS* display window, but can also be called directly from the *Scheme* window.

LISP, SCHEME, AND SIOD

Scheme is a dialect of the *LISP* (List Processing) programming language and is optimized for working with lists of objects, for example, performing the same operation on all channel objects in a list. *Scheme* is a versatile command language whose minimal syntactic structure makes it straightforward to program simple tasks, yet whose sophisticated, recursive nature allows experienced users to program complex operations. The implementation used by *ARS* is *Scheme In One Defun* (*SIOD*), written by Paradigm Associates of Cambridge, MA. *SIOD* is written in *C* and is designed specifically as a command-language interpreter for *C* programs.

This document provides hints for programming simple tasks in *Scheme.* However, because of the access and flexibility provided by the *Scheme* interface, you can create functions that have unintended results. For information about more complex *Scheme* programming, refer to individual texts on *Scheme*; [Fri97] is recommended as a beginning text.

USE OF SCHEME IN ARS

Scheme is used as the user interface to ARS's *C* functionality, either indirectly through the graphical display window or directly through the *Scheme* window. The top-level *C* functions that implement the capabilities needed to manipulate data or display graphical objects are made available to the *Scheme* interface by a technique called binding. The *Scheme* functions that are bound to *C* functions are used like ordinary *Scheme* functions, however, the underlying *C* code is actually executed. These functions are identified in this document as *C* functions.

Scheme functions thus either directly execute a *C* function or are composite *Scheme* functions that consist of calls to other *Scheme* functions. Composite *Scheme* functions are used to extend a basic capability that is provided in *C*. For example, the function `(align-channel)` calls a *C* function that phase-aligns one channel, whereas the function `(align-channels)` is a composite function that phase-aligns a list of channels. The definition of `(align-channels)` can be found in one of the *ARS Scheme* text files, while the definition of `(align-channel)` is embedded in *C* code.

Scheme uses two types of variables, those that are local to the *Scheme* interface and those that are accessible to *C*. The variables accessible to *C* are called CVARs and are set or retrieved by *Scheme* functions bound to *C*. These variables are documented separately from actual *Scheme* variables.

Scheme variables may be defined either as global to the entire *Scheme* interface or local to a single *Scheme* function. Only the global *Scheme* variables are defined in this document.

ARS'S INITIALIZATION OF SCHEME

When *ARS* is first run, the *Scheme* window displays the following (or a similar) message, which lists the *Scheme* files that *ARS* reads during the initialization process:

```
Welcome to SIOD, Scheme In One Defun, Version 2.3
(C) Copyright 1988, 1989 Paradigm Associates Inc.
Initializing scheme extensions
reading file ".../rel/scheme/intrinsic.scm" ... done
reading file ".../rel/scheme/siod.scm" ... done.
reading file ".../rel/scheme/general.scm" ... done.
reading file ".../rel/scheme/math.scm" ... done.
reading file ".../rel/scheme/libpar.scm" ... done.
reading file ".../rel/scheme/ARSdefault.scm" ... done.
reading file ".ARSinit" ...
reading file ".../config/app_config/interactive/ARS/
IDC.scm"... done.
reading file "/home/mist/qaidc2/ARS.load" ... done.
done.
```

The following files define the fundamental *Scheme* functionality common to all the IDC applications that use *Scheme*:

- `intrinsic.scm`
- `siod.scm`
- `general.scm`
- `math.scm`

These *Scheme* files are located in the directory defined by the environment variable `$SCHEMEPATH`. The files `libpar.scm` and `ARSdefault.scm` are also located in this directory. `Libpar.scm` provides interprocess communication support. `ARSdefault.scm` provides *ARS* with its default *Scheme* interface functionality and operating parameter settings. This file should be used as a source of examples when developing custom functions. `ARSdefault.scm` is maintained by *ARS*' developers.

After *ARS* has loaded its core *Scheme* files, it reads the file `.ARSinit` from the user's home directory. This file enables the user to customize functionality, parameter settings, or overrides to the default functionality defined in `ARSdefault.scm`. At the Prototype International Data Centre (PIDC) `.ARSinit` specifies the loading and interpreting of `IDC.scm` and `ARS.load`. The `IDC.scm` file provides custom functionality tailored to the International Data Centre (IDC). This file is located in `$CMS_CONFIG/app_config/interactive/ARS`. Located in the user's home directory along with the `.ARSinit` file, `ARS.load` is a machine-generated file that specifies the time and database for a user's processing session.

Several other files contribute significantly to the *Scheme* interface. The file `ARS.par` is located in the directory `$CMS_CONFIG/app_config/interactive/ARS` along with `IDC.scm` and must be specified on the command line when *ARS* is run. This file, which in turn loads and interprets the common system parameter file, (`$CMS_CONFIG/system_specs/shared.par`), specifies many system pathnames and parameters that are available to the *Scheme* interface as CVARs.

Two X-Window resource files are important for *Scheme* functionality: `app-defaults` and `app-resources`. The `app-defaults` file specified by `$XFILESEARCHPATH` defines the default menu and toolbar items for *ARS* and links these items to functions in `ARSdefault.scm`. This file is maintained by *ARS*' developers. The `app-resources` file specified by `$XAPPLERESDIR` specifies the IDC customization for *ARS* menu and toolbar items. It overrides and adds to definitions in the `app-defaults` file and links the GUI menus and buttons to functions in both `ARSdefault.scm` and `IDC.scm`.

USING SCHEME

After *ARS* initialization, the *Scheme* interpreter echoes a greater-than symbol (>) in the *Scheme* window indicating that it is ready to receive keyboard input. All input to this window is passed to the interpreter for evaluation.[1] If the input is understood by the interpreter, it is evaluated, and its value or output is printed in the window. If it is not understood (for example, if an undefined variable or an incorrect argument type is passed to a function), the interpreter will report an error. In the case of an undefined variable, the following message is displayed:

```
ERROR: unbound variable (see errobj)
```

If the user then types `errobj` at the *Scheme* prompt (>) and presses the <Return> key, the name of the unbound variable will be printed. If an incorrect argument type is passed to a function, the error report will include the argument position and the function name. For example, in the error message:

```
wta (1st) to function-name (see errobj)
```

1. If the (>) symbol does not appear, type `t` (which represents true in *Scheme*; `nil` or `( )` represents false), and press the <Return> key. The interpreter will respond by echoing the `t` and then prompting for more input (>).

`wta` is an acronym for "wrong type argument," and `(1st)` is the parameter position of the offending argument. Again, the user may type `errobj` at the *Scheme* prompt, and the incorrect argument will be reported. Because *Scheme* is interpreted, it is forgiving of errors; *Scheme* simply prints an error message and then prompts for more input.

Variables

In *Scheme,* a variable (often referred to as a symbol) may have either a value or a function bound to it; that is, it may have a definition attached to it. For example, the variable *t* has the non-nil value `t` attached to it to represent true. In the following example, the variable *a-number* is defined and assigned an initial value:

```
> (define a-number 6)
```

After the <return> key is pressed, the value `6` is bound to *a-number*, resulting in a printout of the value `6.000000`. The `set!` function can be used to change a variable's binding. The following example changes the value bound to *a-number*:

```
> (set! a-number 10)
```

The interpreter returns `10.000000`. For the following command,

```
> a-number
```

the interpreter responds with the bound value of the variable: `10.000000`. In this example, *a-number* is not enclosed in parentheses, because it is bound to a simple value (an atom) and can be interpreted directly. The variables *define* and *set!* are bound to functions and can be interpreted only as the first member of a list.

Functions

Scheme distinguishes two fundamental data types: atoms and lists. An atom is a number, symbol, or a text string. A list is a set of elements, atoms, or lists, enclosed in parentheses and separated by spaces. A variable or function name is therefore an atom. The *Scheme* interpreter evaluates an atom to the value bound to it as described in the previous paragraph. The *Scheme* interpreter evaluates a list by treating its first element as a function and the remaining items as parameters (or arguments) to that function. Therefore, in the *Scheme* window, a function is invoked by typing an open parenthesis, the function name, its arguments, and a close parenthesis:

```
> (function parameter parameter ...)
```

The interpreter evaluates the function and returns its value. Functions can return values of any kind: numbers, text strings, lists, and so on. If a list is used as an argument to a function, this list is first evaluated as a function, and its return value is used as the parameter passed to the function containing it. The following examples use the plus function (+):

```
> (+ 2 3)
5.00000
> (+ 2 (+ 3 4))
9.00000
```

In the second example, the expression `(+ 3 4)` is evaluated first, and its result, `7`, is used as the second argument for the outer plus function.

Square brackets are used to denote optional parameters to function calls. When optional parameters are omitted from a function call, *Scheme* will substitute a nil value.

```
> *catch tag form [environment]
```

In *ARS* the compiled *C*-bound functions and those supplied in `ARSdefault.scm`, `IDC.scm`, and the *SIOD* files are all available for use. A user can call any of these functions from the interpreter prompt or incorporate these functions into new, composite functions.

Lists, Evaluation, and ‘

The interpreter attempts to evaluate everything that the user types into it. Lists are evaluated as functions no matter how deeply they are nested. However, *Scheme* provides a way to suppress evaluation. If an expression is prefixed with a single quote (‘), evaluation is suppressed for the entire expression. In an earlier example, the value `10` was bound to the atom *a-number*. When `a-number` was typed, the interpreter evaluated the atom and printed `10.000000`. If `'a-number` is typed instead, evaluation of the atom is suppressed, and the interpreter prints the atom rather than its value. Similarly, if a list is quoted, the interpreter returns the unevaluated list rather than evaluating the list as a function. Quoted lists are commonly used as arguments passed to functions.

Quoting an expression effectively quotes all sub-expressions: evaluation is suppressed up to the matching end parenthesis. Thus `'((((()(()()))))())` is passed as is. Parentheses can get complicated in *Scheme*; for anything other than simple input to the *Scheme* window, an editor is recommended to create the function in a separate text file.

Defining Functions

Functions are defined similarly to the way variables are defined, using the `define` function. The following example defines the function `zoom-and-align`.

```
> (define (zoom-and-align phase)
  (zoom-on-origin)
  (align-channels-on-phase phase))
```

This function will now be available for the duration of the current *ARS* session. It takes one argument, *phase*, which is a phase-name for theoretical arrival time alignment. The function can be called from the interpreter by the following command:

```
> (zoom-and-align "P")
```

As a result, the `(zoom-on-origin)` function is evaluated, then the `(align-channels-on-phase)` function is evaluated with "P" as an argument. Both of these functions are defined in `ARSdefault.scm`. The `(zoom-and-align)` function illustrates how to build composite functions by combining existing functions.

A help string can be added to the newly defined function by calling the `add-to-help-list` function:

```
> (add-to-help-list “(zoom-and-align)”
“Zooms on the selected origin, then aligns on a speci-
fied phase”)
```

NAMING CONVENTIONS

The *ARS Scheme* files collect variable and function definitions to make them available to future *ARS* sessions. They use conventions for naming the *Scheme* functions and variables. In general, names are all lower case, and a dash (-) separates and connects words. The recommended convention for constructing function names is as follows:

```
<verb>-<object>
```

For example,

```
zoom-origin
add-channel
remove-channel
```

This convention may be modified to qualify the verb as follows:

```
<verb>-<indirect-object>-<object>
```

Indirect objects result when the direct action of <verb> on <object> affects a third entity in *ARS*, for example, when modifying a selection list:

```
add-selectlist-channel
remove-window-arrival
```

`<object>` is singular for functions that take an element as an argument and plural for those that take a list. Plural forms should be derived from singular forms by using the `(map)` function. In the special case where all items of type `<object>` are affected, "all" is added to the plural form:

```
add-selectlist-channels-all
```

The following verbs are recommended for use in *ARS Scheme* function names:

`add`	
`remove`	
`create`	
`find`	
`prompt`	(prompts user)
`zoom`	
`compute`	
`set`	(*Scheme* convention, append "`!`" to end)
`get`	(get a new object)
`extract`	(extract an attribute of an object)
`paint`	(for redraws)
`say`	(for functions with output, for example, `say-channels`)
`read`	
`write`	
`show`	
`unshow`	
`sort`	
`associate`	
`disassociate`	

For `set` and `extract`, the direct object is considered to be the attribute that is set or extracted, and the indirect object is the data object from which the attribute is a portion. For example, in `(set-channel-time!)` and `(extract-origin-location)` the direct objects are `time` and `location`, respectively.

Predicate functions make a boolean test and return true or false. Generally, predicate functions that test an attribute use the attribute name with an appended "`?`":

```
null?
eqv?
frozen?
current?
arrival-associated?
arrival-in-time-period?
```

Boolean or predicate variables have the following form:

```
<verb>-<object>-p
```

The recommended syntax for constructing variable names is as follows:

```
<noun>-<adjective>
```

For example,

```
channel-list
```

Data conversion functions often use the -> compound symbol:

```
human-time->epoch-time
```

ARS developers have tried to adhere to these standards; however, standard *Scheme* or *LISP* functions are often exceptions to the naming conventions.

LOADING SCHEME CODE FROM FILES

The `load` function can be used to instruct the interpreter to load and interpret a file containing *Scheme* expressions:

```
> (load '/full/path/to/your/file.scm)
```

Scheme loads the file and signals when it is finished. At startup, *ARS* will automatically load the file `.ARSinit` if the file exists in the user's home directory. The user may add functions to this file or use `(load)` to add functions from another file.

Scheme Intrinsic Functions

This chapter describes the *Scheme* functions standard to the PIDC implementation of *Scheme*. These functions are accessible to all applications that use the *Scheme* interface. This chapter includes the following topics:

- Standard Math Functions
- General Purpose Functions

Scheme Intrinsic Functions

These functions provide the basis for *Scheme* as used by *ARS* and other PIDC applications. Examples are provided in many cases to demonstrate proper use of the functions.

STANDARD MATH FUNCTIONS

The mathematical functions described below are standard to the *Scheme* interface.

+ *num1 num2*

Adds two numerical arguments and returns the result. Errors occur if the arguments are not numbers.

```
> (+ 2 3)
5.00000
```

– *num1 num2*

Subtracts the second numerical argument from the first and returns the result. Errors occur if the arguments are not numbers.

```
> (- 2 3)
-1.00000
```

* *num1 num2*

Multiplies the second numerical argument by the first and returns the result. Errors occur if the arguments are not numbers.

```
> (* 2 3)
6.00000
```

/ *num1 num2*

Divides the first numerical argument by the second and returns the result. Errors occur if the arguments are not numbers.

```
> (/ 2 3)
0.66
```

> *num1 num2*

Compares two numerical arguments and returns `t` if the first argument is greater than the second; otherwise, it returns nil `()`.

```
> (> 1 2)
()
```

```
> (> 2 1)
t
```

< *num1 num2*

Compares two numerical arguments and returns `t` if the first argument is less than the second; otherwise, it returns nil `()`.

```
> (< 1 2)
t
```

```
> (< 2 1)
()
```

>= *num1 num2*

Returns `t` if *num1* is greater than or equal to *num2*; otherwise, it returns nil `()`.

<= *num1 num2*

Returns `t` if *num1* is less than or equal to *num2*; otherwise, it returns nil `()`.

`max` *num1 num2*

Returns the maximum of the two values passed as *num1* and *num2*.

`min` *num1 num2*

Returns the minimum of the two values passed as *num1* and *num2*.

The following *C* math library functions have a *Scheme* interface. For more information about functions, refer to `(man -s3M Intro)` in the *IDC Software Man Pages* [IDC6.4Rev1].

`cos` *argument*

Applies the cosine function to an *argument*. Units are in radians.

`sin` *argument*

Applies the sine function to an *argument*. Units are in radians.

`tan` *argument*

Applies the tangent function to an *argument*. Units are in radians.

`acos` *argument*

Applies the arccosine function to an *argument*.

`asin` *argument*

Applies the arcsine function to an *argument*.

`atan` *argument*

Applies the arctangent function to an *argument*; returns a value between -pi/2 and pi/2.

`atan2` *argument1 argument2*

Applies the arctangent function to *argument1* and *argument2*; returns a value between -pi and pi.

`exp` *argument*

Returns the value of $e^{argument}$.

`expm1` *argument*

Returns the value of ($e^{argument}$) – 1 for even the smallest argument; for example, `(expm1 .0003)` will be greater than zero.

`exp2` *argument*

Returns the value of $2^{argument}$.

`exp10` *argument*

Returns the value of $10^{argument}$.

`log` *argument*

Returns the natural log, ln (*argument*).

`log1p` *argument*

Returns the value of ln (1 + *argument*), for even a small *argument*.

`log2` *argument*

Returns $\log_2$ *argument*.

`log10` *argument*

Returns $\log_{10}$ *argument*.

`pow` *argument1 argument2*

Returns the value of $argument1^{argument2}$.

`pi`

Returns `3.14159265358979323846`.

`sqrt` *argument*

Applies the square-root function to an *argument*.

`fabs` *argument*

Applies the absolute-value function to an *argument*.

`cosh` *argument*

Applies the hyperbolic-cosine function to an *argument*.

`sinh` *argument*

Applies the hyperbolic-sine function to an *argument*.

`tanh` *argument*

Applies the hyperbolic-tangent function to an *argument*.

`acosh` *argument*

Applies the hyperbolic-arccosine function to an *argument*.

`asinh` *argument*

Applies the hyperbolic-arcsine function to an *argument*.

`atanh` *argument*

Applies the hyperbolic-arctangent function to an *argument*.

`fmod` *x y*

Returns *x* mod *y*. For example, `(fmod 10.1 3)` returns `1.100000`.

`floor` *x*

Returns the largest integer less than *x*. For example:

```
> (floor -2.3)
-3.000000
```

```
> (floor 2.3)
2.000000
```

`ceil` *x*

Returns the smallest integer larger than *x*. For example:

```
> (ceil -2.3)
-2.000000
```

```
> (ceil 2.3)
3.000000
```

`rint` *x*

Returns *x* rounded to the nearest integer. For example:

```
> (rint -2.3)
-2.000000
```

```
> (rint 2.3)
2.000000
```

GENERAL PURPOSE FUNCTIONS

The general purpose functions described in this section are standard to the *Scheme* interface. They provide the basic functionality used in writing *Scheme* programs.

`abort`

Terminates *ARS* (or calling application). Any unsaved data may be lost. This function should be used with extreme caution.

`and` *expression1 expression2*

Predicate function that returns the logical "and" of the two expressions.

`append` *list1 list2 list3 ... listN*

Returns a list containing the elements of *list1–listN* in the same order as passed. The arguments must be lists. The passed lists are not modified.

```
> (define a (list 1 2 3))
(1.000000 2.000000 3.000000)

> (define b (list 3 4))
(3.000000 4.000000)

> (define c (list 5 6))
(5.000000 6.000000)

> (append a b c)
(1.000000 2.000000 3.000000 4.000000 5.000000 6.000000)
```

`apply` *function list*

Applies a *function* to the arguments collected in *list*.

```
> (apply '* (list 1 2))
(6.000000)
```

`apply-to-all` *function list*

Applies a *function* that takes a single argument to every element in the *list*. This function is similar to `map` except it does not return the list of values; this function is more efficient if the return list is not needed.

`assq` *arg1 assoc-list*

Takes an item and an association list as its arguments. The association list contains symbol-value pairs. For example, the following association list,

```
(define assoc-list
(list (cons 'k1 'v1) (cons 'k2 'v2) (cons 'k3 'v3)))
```

returns

```
((k1 . v1) (k2 . v2) (k3 . v3))
```

`assq` returns the associated pair in the association list for the given item, so

```
> (assq 'k1 assoc-list)
```

returns

```
(k1 . v1)
```

`backquotify` *arg*

Evaluates an argument, *arg*, and returns it as quoted, allowing its evaluation to be postponed.

```
> (define a (list 1 2 3))
(1.000000 2.000000 3.000000)

> a
(1.000000 2.000000 3.000000)

> (backquotify a)
(quote (1.000000 2.000000 3.000000))
```

`begin` *body*

Identifies the beginning of a series of subexpressions. Each of the sub-expressions is evaluated consecutively in the order that it appears, and the value of the last subexpression is returned as the value of the `begin` expression.

```
(if (eqv? "My string" string-holder)
     (begin
          (print "My string")
          (print " does equal")
          (print " the value of")
          (print " string-holder"))
(print "The two strings do not match"))
```

`calculate` *function argument1 argument2*

Returns the value of the *function*, represented as a string, applied to *argument1* (and *argument2,* if necessary). This function maps to the *C* math library functions.

`car` *arg-list*

Returns the first item in the argument list *arg-list*.

```
> (car (list 1 2 3))
```

returns the atom

```
1.000000
```

`*catch` *tag form* `[`*environment*`]`

Evaluates *form* and returns its value unless a `(*throw` *tag value*`)` is executed, in which case it returns *value*. This function is always used in conjunction with `*throw`. The following example uses `*throw` and `*catch`:

```
(define (multiply-list x)
  (*catch 'zero-value
  (sub-mult x)))
```

```
(define (sub-mult y)
  (cond ((null? y) 1)
  ((eqv? (car y) 0)
  (*throw 'zero-value 0))
  (t (* (car y) (sub-mult (cdr y))))))
```

So,

```
> (multiply-list '(2 3 4 5))
```

returns

```
(120)
```

but,

```
> (multiply-list '(2 0 3 4 5))
```

returns

```
(0)
```

It returns the value immediately upon detecting the zero value rather than multiplying the entire list.

`cdr` *arg-list*

Returns a list consisting of all items in the argument list with the exception of the first item.

```
> (cdr '(1 2 3))
(2 3)
```

In this example, a quoted list is used as `cdr`'s sole argument. The `'` character is synonymous with `quote`; `quote` delays evaluation so the user can access the symbol instead (see `quote`).

```
> (cdr (cons 1 2))
2.000000
```

cond *<condition1 consequent1> <condition2 consequent2>...*
<conditionN consequentN>

Evaluates conditions by implementing a "case analysis." Only the first *condition* evaluated as true will have its *consequent* evaluated. Typically the last *condition* is specified as the truth value; this accounts for the default case. For example:

```
(define (find-sole-arrival)
    (cond ((null? (say-selected-arrivals))
    (begin (say-info (list "Choose an arrival!"))
    (error "Must choose an arrival")))
    ((not (eqv? 1 (length (say-selected-arrivals))))
    (begin (say-info (list "Choose only one arrival!"))
    (error "Only choose one arrival")))
    (t (car (say-selected-arrivals)))))
```

Here, the conditional expression has three conditions, namely:

1) `(null? (say-selected-arrivals))`

2) `((not (eqv? 1 (length (say-selected-arrivals)))))`

and the default condition

3) `(t (car (say-selected-arrivals)))`

`cons` *arg1 arg2*

Constructs a list from its two arguments.

```
> (cons 1 '(2))
(1 2)
```

The empty list is denoted as `()` and is also referenced as nil. For example:

```
> (cons 1 '())
(1)
```

This function yields the same result as

```
> (cons 1 ())
```

because the second argument, the empty list, is nil. If the second argument to `cons` is not a list, the function returns a dotted pair, not an ordinary list. For example:

```
> (cons 1 2)
(1 . 2)
```

Users can perform operations on the dotted pair similarly to the way they would on normal list.

```
> (set! j (cons 1 2))
(1 . 2)

> (car j)
1.00000

> (cdr j)
2.00000
```

`copy-list` *list*

Returns a copy of the *list* passed in.

`define` *arg-list*

Defines a symbol and assigns it a value, which can be a number, a list, or a function. For example:

```
> (define a 2)
2.000000
```

```
> a
2.000000
```

```
> (+ 1 a)
3.000000
```

`env-lookup` *arg env*

Uses the environment, *env*, to look up the symbol *arg*. For example, if *env* is defined as follows:

```
> (define env   (let ((x 5 (y 4)) the-environment)))
```

the new values of *x* and *y* are imported into the environment for *env*:

```
> (env-lookup 'x env)
(5.000000 4.000000)
```

Using the environment is an advanced feature of *Scheme* and is beyond the scope of this document. See `the-environment` for an example of using environments.

`epoch-time->human-time` *time*

Converts an epoch *time* (number of seconds since the epoch, January 1, 1970) to a readable, human time.

The following example returns the time in the readable, human format "1990-03-06 13:43:26":

```
> (epoch-time->human-time 636731006.00)
```

`epoch-time->yyyydoy` *epoch-time*

Calculates the Julian date of the specified UNIX epoch time. For example:

```
> (epoch-time->yyyydoy 300000000)
1979186.000000
```

`eq?` *arg1 arg2*

Predicate function that returns `t` if the two arguments are identical, that is, they point to the same address in memory. For example:

```
> (eq? 'a 'a)
t
```

If the user creates a symbol with the same value, the arguments are not equivalent:

```
> (set! j 1)
1.000000

> (set! k 1)
1.000000

> (eq? j k)
()
```

`eqv?` *arg1 arg2*

Predicate function that returns `t` if the value of the first argument is equivalent to the value of the second argument. This example uses the values of *j* and *k* as defined in the previous example

```
> (eqv? j k)
t
```

`error` *error-message* `[`*symbol*`]`

Displays the error message, *error-message*, sets *errobj* to be bound to *symbol*, and performs a `longjmp()` out of the encompassing *Scheme* code. This function is implemented in *C*.

```
(define (delete-arrivals)
 (if (null? (say-selected-arrivals))
  (begin (say-info (list "Select
   arrivals to delete."))
   (error "Select arrivals to delete."))
(map delete-arrival (say-selected-arrivals))))
```

`eval` *expression [environment]*

Evaluates *expression* in the context of *environment*. If *environment* is omitted or set to nil, the user's UNIX environment is used by default. `eval` then simplifies to evaluating *expression*. For example:

```
> (set! fn '(+ 1 2))
(+ 1.000000 2.000000)
> (eval fn)
3.000000
```

Users can also define their own *environment* and evaluate *expression* referencing this *environment*. Using the environment is an advanced feature of *Scheme* and is beyond the scope of this document. See `the-environment` for an example of `eval` using *environment*.

`extract-par-value` *par-str par num-p vec-p*

Sets up libpar string *par_str* and gets the value for the variable *par* using the format specified in *num-p* and *vec-p*; then libpar is closed. If *num-p* is `t`, the return value will be a number. For `nil` it will be a string. If *vec-p* is a `t`, the return value will be a list of all the values for the parameter. If it is `nil`, the value will be an atom. Double quotes are placed around a string with single quotes. For example:

```
> (extract-par-value  "aaa=123 bbb=0.123" "aaa" t nil)
123.000000
> (extract-par-value  "aaa=123 bbb=0.123" "aaa" nil nil)
"123"
> (extract-par-value  "aaa=123 bbb=0.123" "aaa" t t)
(123.000000)
```

`file-exists?` *file*

Checks if *file* exists and if so, returns `t`. For example:

```
> (file-exists? "/etc/passwd")
t
```

`find-file` *pathlist file suffix*

Searches a *pathlist* (directories) for a *file* with a *suffix* and returns a string that is the completed path to the *file* or nil if the *file* is not found.

```
>    (find-file    "/vobs/rel/pidc_6.2/config/earth_specs/TT/
iasp91" "iasp91" "Pn")
"/vobs/rel/pidc_6.2/config/earth_specs/TT/iasp91.Pn"
```

`find-member` *argument list*

Searches for an *argument* in a *list* and, if found, returns the remainder of the *list*; otherwise, it returns `nil`.

`for-each` *function arg-lists*

Takes a list containing a *function* and a list of arguments and applies the function onto every argument on the `cdr` of the list.

```
> (for-each print '(a b c))
(a b c)
t
```

```
> (for-each print '(a b c) '(d e f))
(a b c)
(d e f)
t
```

`format-num` *format-string fnum*

Uses the *C* `sprintf` function to create a string with the number *fnum* using the format specified by *format-string*. For example:

```
> (format-num "%5.3f" 3.1)
"3.100"
```

```
> (format-num "%8.3f" 3.1)
"   3.100"
```

```
> (format-num "%-8.3f" 3.1)
"3.100   "
```

`gc` [*status*]

Executes the garbage collection function. A non-nil *status* argument causes timing statistics to be displayed. Garbage collection occurs automatically, users need not explicitly perform this function.

`gc-status` [*status*]

Controls verbosity for garbage collection of unused objects. Without any arguments, `gc-status` reports the current level of garbage collection verbosity (either verbose or silent) as well as the number of allocated versus freed cells. With a non-nil argument, it activates verbosity and reports the number of cells in use as well as the amount of time it took to collect the garbage. With a nil argument, verbosity is turned off. Examples:

```
> (gc-status)
garbage collection silent
13080 allocated 36920 free
()

> (gc-status 1)
garbage collection verbose
13083 allocated 36917 free
()

[starting GC]
[GC took 0.2 cpu seconds, 1 cells collected]

> (gc-status nil)
garbage collection silent
13085 allocated 36915 free
()
```

`get-cvar-number` *variable*

Returns a number representing the value of an internal *C variable*. The following example returns a number representing the start time of data read by *ARS*:

```
(get-cvar-number "start_time")
```

`get-cvar-string` *variable*

Returns a string representing the value of an internal *C variable*. The following example returns a string representing the start time of data read by *ARS*:

```
(get-cvar-string “start_time”)
```

The following example returns a string representing the name of the database read by *ARS*:

```
(get-cvar-string “database”)
```

`getenv` *str*

Runs the *C* `getenv` library call to determine the environment setting for the variable specified in *str*. If no such setting exists, a null string is returned. The function returns a string. For example:

```
> (getenv "HOME")
"/home/mist/qaidc"

> (set! usrname (getenv "USER"))
"jim" (and usrname is set)

> (getenv "XYZ")
""
```

`human-time->epoch-time` *human-time-string*

Converts a readable *human-time-string* to epoch-time (number of seconds since the epoch, January 1, 1970). For example:

```
(human-time->epoch-time “6/9/1990 15:00”)
644943600.000000
```

Fractional seconds are not permitted.

`if` *conditional-form consequent* `[`*alternative*`]`

This conditional form only has two cases: The syntax is (`if` *condition consequent alternative*) or (`if` *condition consequent*). In the first case, if *condition* is true, the value of *consequent* is returned as the value of the expression; if the condition is false, the value of *alternative* is returned as the value of the expression. In the second case, the *alternative* is not present; if the condition is false, nil is returned.

`lambda` *arg body*

Scheme construct that describes a function. This expression is entered as a list; the first element must be the symbol `lambda`, the second element must be an argument list, *arg*, and the remaining elements constitute the *body* of the function. Thus the list `(lambda (x) (+ x 5))` specifies a function that adds `5` to `x`.

This expression is used to define a function within the definition of another function. For example, recall that the `map` function applies a function to all elements of a list. The `map` function can be used to add 5 to all of the numbers in a list. Rather than define an external "`add 5`" function, a `lambda` expression can be used to specify the new function inside `map`:

```
> (map (lambda (x) (+ x 5)) '(1 2 3 4 5))
(6.000000 7.000000 8.000000 9.000000 10.000000)
```

The `(+ x 5)` function is applied to each number by substituting `x`.

`last` *list*

Returns a list with the last item in the *list*.

```
> (define a (list 1 2 3 4 5 6))
(1.000000 2.000000 3.000000 4.000000 5.000000 6.000000)

> (last a)
(6.000000)
```

`length` *list*

Returns the number of elements in a *list*.

`let ((`*var1 val1*`) (`*var2 val2*`)...(`*varn valn*`))` *body*

Provides variables that are lexically scoped. The variables exist only within the confines of the `let` expression. An instance of a local variable *var* is created with value <*val*>, which can be used within *body*. An unlimited number of variable/value pairs can be used at one time. For example:

```
> (let ((a 2) (b 3))
(+ a b)
5.00
```

```
> a
ERROR: unbound variable (see errobj)
```

The variables are independent of each other. Consequently, the value of *varn* cannot depend on a preceding variable binding.

```
> (let ((a 2) (b (+ a 1))) (print b))
ERROR: wta(1st) to + (see errobj)
```

A variable may be overloaded, that is, redefined for a different scope; after the variable is outside of the scope, it reverts back to its previous value.

`list` *arg1 arg2 ...argn*

Returns the *arg*uments as a list.

```
> (list 'a 'b 'c)
(a b c)
```

`load` *file*

Opens and reads the file specified by the full pathname *file*. If the function is unsuccessful at either opening or reading *file,* an error message appears.

```
> (load '/data/alvis/functions.scm)
loading /data/alvis/functions.scm done.
()
```

```
> (load '/no_such_file)
loading /no_such_file
loading /no_such_file: No such file or directory
ERROR: could not open file
```

`make-list-filter` *predicate-function*

Takes a *predicate-function* and creates a new function that takes a list argument. When the new function is evaluated, it returns a new list with elements in the original list for which *predicate-function* evaluated as true.

`map` *function list*

Applies a single argument *function* to every element in *list* and returns a list of the results.

```
> (map sqrt '(1 4 9))
(1.000000 2.000000 3.000000)
```

`nconc` *arg1 arg2*

Sets the *cdr* of the first argument *arg1* to point to *arg2*. This creates a copy of *arg2* and does not point to the same address.

```
> (define a (list 1 2))
(1.000000 2.000000)

> (define b (list 3 4))
(3.000000 4.000000)

> (nconc a b)
(1.000000 2.000000 3.000000 4.000000)

> a
(1.000000 2.000000 3.000000 4.000000)

> b
(3.000000 4.000000)
```

`no-op`

No operation; acts only as a place holder. This function can take as many arguments as necessary.

`not` *expression*

Predicate function that negates the value of the predicate *expression*.

```
> (not t)
()
```

```
> (not ())
t
```

```
> (if (not (eqv? 10 20))
(print "10 is not equal to 20")
print "Defying logic, 10 IS equal to 20"))
"10 is not equal to 20"
()
```

`null?` *symbol*

Predicate function that returns true (`t`) if the value of the *symbol* is nil; otherwise, it returns nil `()`.

```
> (define a 2)
2.000000
```

```
> (null? a)
()
```

```
> (set! a ())
()
```

```
> (null? a)
t
```

`number?` *arg*

Predicate function that returns `t` if *arg* is a number; otherwise, it returns `nil`. Examples:

```
> (number? 10)
t

> (define j 10)
10.000000

> (number? 'j)
()

> (number? j)
t

> (number? "j")
()
```

`num-to-string` *num*

Converts the number *num* to a character string.

```
> (num-to-string 1123)
"1123.000"

> (define a 14)
14.000000

> (num-to-string a)
"14.000"
```

`oblist`

Returns the list of all symbols. Directly invoking this function causes the return list to be added to the defined symbols (entering `oblist` twice will cause twice as many symbols to be displayed).

`on-list?` *atom list*

Returns an *object* if that *atom* is in the *list*; otherwise, it returns `nil`.

`or` *expression1 expression2*

Returns the logical OR of the two expressions.

`popen` *command*

Opens a pipe to a shell *command*, and returns the output of the *command* as a *LISP* string. The *command* calls the UNIX `popen(2)` command to execute the shell *command*. Only the first line of the *command*'s output is returned, and the trailing new line is stripped. The following example returns a string containing the system date and time:

```
(popen “date”)
```

`quit`

Acts much like `*throw` except it discards any return value; it "quits" the current evaluation.

`quote` *arg*

Returns an unevaluated expression (not its value). `quote` may also be referred to as the symbol `'`.

```
> (define a 3)
3.000000

> (quote a)
a

> (quote (+ 2 3))
(+ 2.000000 3.000000)

> ‘(+ 2 3)
(+ 2.000000 3.000000)
```

`quote-string` *str*

Takes a *string* and returns it as a new string enclosed in quotes for use by programs that require quoted strings (such as `send-ipc-message`).

```
> (quote-string "abc")
""abc""
```

If the input string is nil, `nil` is returned.

`pair?` *arg*

Predicate function that returns `t` if its *arg*ument is a cons cell; otherwise, it returns `nil`.

`print` *form*

Prints its evaluated argument. Examples:

```
> (print t)
t
()
```

```
> (print "String")
"String"
()
```

```
> (print find-sole-arrival)
#<CLOSURE ()
(cond ((null? (say-selected-arrivals))
(begin (say-info (list "Choose an arrival!"))
(error "Must choose an arrival")))
((not  (eqv?  1.000000  (length  (say-selected-arrivals))))
(begin (say-info (list "Choose only one arrival!")) (error
"Only choose one arrival")))
 (t (car (say-selected-arrivals))))
()
```

`remove-duplicates` *list*

Removes duplicate atoms from *list*. Remaining atoms are returned in a list that is in reverse order from the original *list*.

`remove-list-object` *list object*

Builds and returns a new list from the elements of the argument *list*, excluding the element specified by *object*. The original list is not modified.

`replace` *before after*

Replaces the *before* argument with a duplicate of the *after* argument (the *car* value is set to the *car* of *after*, and the *cdr* value is set to the *cdr* of *after*).

`reverse` *list*

Returns the argument *list* as a list in reverse order.

`say-cvars`

Lists the known CVARS in alphabetical order with their current settings.

`say-time-now`

Returns the current epoch time (seconds since January 1, 1970).

`set!` *symbol value*

Sets the *symbol* to the *value*. The value could be another symbol, group of symbols, or any *Scheme* expression.

```
(set! a "a")
(set! a (cons 1 2))
```

`set-car!` *arg-list arg2*

Replaces the first item in a list with the value of *arg2* and returns the replacement item. For example, if the symbol `j` contains the list `(1 2 3)`,

```
> (set-car! j 4)
```

sets `j` to the list:

```
(4.00 2.00 3.00)
```

`set-cdr!` *arg-list arg2*

Replaces all but the first item of the list with the value of *arg2*. If the *arg2* is not a list, the result is a dotted pair.

`set-cvar!` *variable value*

Sets the *value* of an *ARS* internal *C variable*. Both parameters must be strings. For example:

```
(set-cvar! “filter_parameters” “1.5 2.5 3 BP 0”)
```

`set-symbol-value!` *symbol value* `[`*environment*`]`

Sets the *symbol* to be bound to the *value* within the *environment*.

```
> (define a 10)
10.000000

> a
10.000000

> (set-symbol-value! 'a 12)
12.000000

> a
12.000000
```

If the *environment* variable is nil or omitted, then *environment* defaults to the user's UNIX environment, and `set-symbol-value` behaves like `set!`. Using the environment is an advanced feature of *Scheme* and is beyond the scope of this document. See `the-environment` for an example of using environments.

`space-append` *list*

Takes a list of double-quoted strings and returns a single string separated by spaces.

`string-append` *string1 string2 .... stringN*

Concatenates the string arguments to form a single string. For example:

```
> (string-append
"This " "is " "a " "string" "but" "don't" "forget "
"spaces")
"This is a stringbutdon'tforget spaces"
```

`string?` *arg*

Predicate function that returns `t` if *arg* is a string; otherwise, it returns nil `()`.

```
> (string? "This is a string”)
t
```

```
> (string? 9)
()
```

```
> (define value "This is a string")
"This is a string"
```

```
> (string? value)
t
```

`string=` *string1 string2*

Predicate function that returns `true` if *string1* is equal to *string2*.

```
> (string= value "This is a string")
t
```

```
> (string= value "This isn't the string")
()
```

`string<` *string1 string2*

Predicate function that returns `t` (true) if *string1* is lexically less than *string2*. For example:

```
> (string< value "This is a string")
()
```

```
> (string< value "This isn't the string")
t
```

```
> (string< value "---")
()
```

`string>` *string1 string2*

Predicate function that returns `t` (true) if *string1* is equal to *string2*.

```
> (string> value "This is a string")
()
```

```
> (string> value "This isn't the string")
()
```

```
> (string> value "---")
t
```

`string-length` *str*

Returns the number of characters in *str*. For example:

```
> (string-length "abcdef")
6.000000
```

```
> (string-length "")
0.000000
```

`string-ref` *str idx*

Returns a string containing the character in *str* at index *idx*. The first character is at index 0. For example:

```
> (string-ref "abcdef" 4)
"e"
```

```
> (string-ref "abcdef" 0)
"a"
```

```
> (string-ref "abcdef" 7)
() ERROR: string-ref: index exceeds string length (see
errobj)
```

```
> (string-ref "abcdef" -1)
() ERROR: string-ref: value out of range (see errobj)
```

`string-rsearch` *str char-idx*

Returns the substring of *str* that starts at the character given by *char-idx*, searching from the end of *str*. For example:

```
(string-rsearch "abcdef" "c")
"cdef"

(string-rsearch "abcdef" "z")
()

(string-rsearch "abcdcef" "c")
"cef"
```

`string-search` *str char-idx*

Returns the substring of *str* that starts at the character given by c*har-idx*, searching from the start of *str*. For example:

```
(string-search "abcdef" "c")
"cdef"

(string-search "abcdef" "z")
()
```

`string-to-num` *string*

Converts the *string* into a number.

```
> (string-to-num "10")
10.000000

> (define a "2")
"2"

> (string-to-num a)
2.000000
```

`substring` *str pos1 pos2*

Creates a substring from *str* with the characters from position 1 (*pos1*) to position 2 (*pos1*). The character indices start from zero. *pos1* is inclusive, but *pos2* is exclusive. Both *pos1* and *pos2* must be non-negative and less than the *strlen* of *str*. For example:

```
(substring "abcdefghi" 2 4)
"cd"

(substring "abcdefghi" 0 4)
"abcd"

(substring "abcdefghi" -1 4)
() ERROR: substring: value out of range (see errobj)

(substring "abcdefghi" 0 9)
"abcdefghi"

(substring "abcdefghi" 0 10)
() ERROR: substring: value out of range (see errobj)
```

`symbol?` *arg*

Predicate function that returns `t` (true) if the argument, *arg*, is a symbol; otherwise, it returns nil `()`. For example:

```
> (symbol? t)
t

> (define j 10)
10.000000

> (symbol? j)
()

> (symbol? 'j)
t

> (symbol? 10)
()
```

`symbol-bound?` *symbol* `[`*environment*`]`

Predicate function that returns `t` (true) if the *symbol* is bound to a value in the given environment. If *environment* is nil or absent, the global UNIX environment is used.

```
> (symbol-bound? t)
t
```

In the following example, the *symbol* `a` has never been defined:

```
> (symbol-bound? 'a)
()
```

Using the environment is an advanced feature of *Scheme* and is beyond the scope of this document. See `the-environment` for an example of using environments.

`symbol-value` *symbol* `[`*environment*`]`

Returns the value that the *symbol* is bound to in the *environment*. If *environment* is nil or absent, the global environment is used.

```
> (symbol-value t)
t
```

```
> (define a 10)
10.000000
```

```
> (symbol-value 'a)
10.000000
```

Using the environment is an advanced feature of *Scheme* and is beyond the scope of this document. See `the-environment` for an example of using environments.

`symbolconc` *arg1 arg2 arg3...argn*

Takes the arguments and returns their symbols concatenated together.

```
> (symbolconc 'x 'y)
xy
```

`system` *str*

This function is analogous to the *C* system call: it forks and executes a shell, then executes the command in the string (*str*). *Stdout* (if any) from the command is directed to the *ARS Scheme* window. For example:

```
(system "date")
```

`the-environment [`*arg env*`]`

Defines the environment or a pseudo scope in which objects exist.

```
(define x 100)
(define y 100)

(define env
 (let ((x 5)
 (y 4))
 (the-environment)))
```

This imports the new values of *x* and *y* into the environment for *env*.

```
(+ x y)
200.000000

(eval '(+ x y) nil)
200.000000

(eval '(+ x y) env)
9.000000

(+ x (eval 'y env))
104.000000
```

The use of environments is an advanced feature of *Scheme* and is beyond the scope of this document.

`*throw` *tag value*

Returns the innermost catch expression marked *tag* with *value*. This function is always used in conjunction with `*catch`.

See `*catch` for an example.

Scheme Variables

This chapter describes the *Scheme* variables and includes the following topics:

- Simple Variables
- List Variables
- C Variables

Scheme Variables

The variables local to the *Scheme* interface are divided into two groups: simple variables and lists. Simple variables hold a single entity such as a numeric value or a string of characters. Lists hold multiple entities. Variables that are identified as "overriding the standard" are redefined in `IDC.scm` to override the definition defined in `ARSdefault.scm`. If a variable is only defined in `IDC.scm` it is identified as not overriding the standard.

This chapter also describes CVARs, the variables accessible by both the *Scheme* and *C* segments of the *ARS* software.

SIMPLE VARIABLES

`*additional-elements*`

Lists non-refsta/non-array channel elements to display at selected stations. This variable provides a list in the form of (*el1 el2 ...*) to add elements *el1 and el2* to displayed channels.

Called by `(show-best-channels)`.

`*allowable-waveform-duration*`

For existing waveform, checks if waveform data is in the interval 'theoretical P time' through 'theoretical P time' + `*allowable-waveform-duration*`, then displays the channel.

Called by `(show-best-channels)`.

`alpha-list-help`

Specifies the Help information displayed in the alpha list through the Help button. The text is composed of several appended strings so that the *Scheme* input buffer does not overflow. Text should be left-justified and should be no more than 80 characters per line.

`alphalist-chan-sta-background-color`

Specifies the background color to use in the alpha list for channels.

`alphalist-defining-background-color`

Specifies the background color to use in the alpha list for channels if an attribute is defining (for example, *magdef, slodef, azdef*).

`alphalist-detect-phase-background-color`

Specifies the background color used in the alpha list for detections.

`alphalist-generic-background-color`

Specifies the default background color for the alpha list.

`alphalist-orig-orid-background-color`

Specifies the background color to use in the alpha list for origins.

`area-of-interest-table`

Specifies the path to the area-of-interest table.

`arrival-on-duration`

Specifies the duration, in seconds, below which the arrival labels are automatically displayed.

`arrival-window-width`

Specifies the half-width of the new window, in seconds, when the `zoom-on-arrival` function is called.

Initial value: 30

Calling object: `zoom-on-arrival`

Overrides standard.

`*blockage-grid-dir*`

Specifies the path to the blockage files for the hydroacoustic raypaths.

Initial value:
`(string-append (getenv "CMS_CONFIG") "/earth_specs/BLK_OSO")`

Calling object: `check-associated-hydro-blockage`

Overrides standard.

`broken-associated-origin-color`

Specifies the color to paint arrivals associated with broken origins.

`broken-color`

Specifies the color to paint broken objects. Color is a quoted string containing a valid color name, for example: “`blue`” or “`sky blue`.” Valid colors may be obtained by running *showrgb*, a standard *X* program.

`channel-color`

Specifies the color to paint channel objects.

`*check-had-error*`

Temporary variable used for collecting error text.

Initial value: `nil`

Calling objects:
`check-origin-list`
`add-origin-error`
`add-assoc-error`

Does not override standard.

`*check-text*`

Temporary variable used for collecting text.

Initial variable: `nil`

`*component-order*`

Specifies the preferred order for component display; usually set to be the same `chan-sort-list`.

Called by `(show-best-channels)`.

`*crustal-thickness*`

Specifies the nominal thickness, in kilometers, of the crust. This number is used to test for teleseismic phases at regional distances that should not occur for crustal events.

`current-color`

Specifies the color to paint current completed objects.

`data-on-duration`

Specifies the duration, in seconds, below which waveforms are automatically displayed.

Initial value: `(*60 60)`

Calling object: `show-display-detail`

Overrides standard.

`default-color`

Specifies the default color with which to draw objects.

`*default-hydro-component*`

Specifies the default channel to use for hydroacoustic channels.

`*default-hydro-filter*`

Specifies the default filter to use for hydroacoustic channels.

`*default-magnitude*`

Specifies the default magnitude to assume for computing detection probabilities in `(show-best-chans)` if the existing magnitude is invalid.

Called by `(show-best-channels)`.

`default-phase`

Specifies the default phase to be used by some functions.

Initial value: `"P"`

Calling objects:
`save-and-run-EvLoc`
`verify-orid-selected`
`set-default-phase`
`rename-and-associate-P-to-origin`
`beam-p-phase`

`default-time-def`

Controls the default time defining status of added arrivals.

Initial value: `"d"`

Calling objects:
`create-arrival-with-phase`
`set-default-time-def-on`
`set-default-time-def-off`

Does not override standard.

`derived-channel-color`

Specifies the color to paint derived channel objects. A CVAR of the same name exists in `ARSdefault.scm`.

Initial value: `"black"`

Calling object: `say-color-by-state-object`

Overrides standard.

`*event-confirmation-threshold*`

Specifies the threshold for weighted count using weights in `*phase-weights-for-event-definition*`.

`expand-factor`

Specifies the factor to be used by the `expand-window` function.

Initial value: `0.5`

Calling object: `expand-window`

Overrides standard.

`frozen-color`

Specifies the color to paint frozen objects. A CVAR of the same name exists in `ARSdefault.scm`.

Initial value: `"forestgreen"`

Calling objects:
`paint-frozen-object`
`say-color-by-state-object`

Overrides standard.

`*hydro-station-filters*`

Specifies a list of filters to use for hydroacoustic processing.

`language`

Specifies the language to use for prompts. This variable is used by the `is-lang?` function to check for a match. `language` is an atom, for example,

`“american”`

Currently only american is supported.

`*large-event-primary-phase-count*`

Specifies the number of primary phases in an event to flag it as a large event.

`*last-error-origin*`

Stores the last error encountered by `(add-origin-error)` or `(add-assoc-error)`.

Initial value: `nil`

Calling objects:
`add-origin-error`
`add-assoc-error`

`locator-help`

Specifies the help string that is displayed in the locator interface using the "Help" button. The text is composed of several strings appended together so that the *Scheme* input buffer does not overflow. Text should left justified and no more than 80 characters per line.

`magnitude-corr-directory-prefix`

Specifies the pathname to prepend to magnitude attenuation files; the pathname is a string of the form "*/dir/subdir/*."

Initial value:
```
(string-append (getenv "CMS_CONFIG") "/
earth_specs/MAG/atten/atten")
```

Calling objects:
```
show-best-chans
read-default-travel-time-tables
```

Overrides standard.

`*max-hydro-origin-depth*`

Specifies the maximum origin depth for hydro station display.

`*maximum-window-duration*`

Specifies the maximum seconds to display hydroacoustic channels on screen.

`*min-tele-delta*`

Specifies the minimum distance for teleseismic phases to be allowable. (Not to be confused with `*regional-distance-cutoff*`).

`*minimum-alpha-stations*`

Specifies the minimum number of allowable alpha stations for a confirmed event.

`*minimum-magnitude*`

Specifies the minimum magnitude to assume for computing detection probabilities in `(show-best-chans)` if the existing magnitude is invalid or smaller than this value.

Called by `(show-best-channels)`.

`min-zoom-duration`

Specifies the minimum duration that is accepted by `zoom-on-origin`.

`*newline*`

Contains a newline string for use in writing error messages.

Initial value: `" "`

Calling objects:
`add-origin-error`
`add-assoc-error`
`check-origin-list`

Does not override standard.

`not-current-associated-arrival-color "purple"`

Specifies the color to paint arrivals that are not associated with an origin.

`not-current-associated-origin-color`

Specifies the color to paint arrivals that are associated with noncurrent origins (their location hypothesis is not current with associated data).

`*phase-weights-for-event-definition*`

Specifies the weighting given to each phase attribute. Each element has the form (*phase-type station-type time-weight az-weight slow-weight*). (See Table 3 of [Bow95].)

`*pkkp-distance-cutoff*`

Specifies the maximum delta to use for PKKP phases when searching for special stations for an event.

Called by `(show-best-channels)`.

`pre-origin-time`

Specifies the time to be displayed before the first theoretical arrival when aligning traces or running `zoom-on-origin`.

Initial value: `50.0`

Calling objects:
`show-and-align-waveform-arrival-channels`
`say-origin-interval`
`zoom-on-origin`
`zoom-on-origin-old`

Overrides standard.

`pre-stassoc-time`

Specifies the time to be displayed before the first theoretical arrival when aligning traces or running `zoom-on-stassoc`.

`*prob-atten-file*`

Specifies the path to the attenuation file used by the `libprob` library for computing detection probabilities.

`*prob-noise*`

Specifies the characteristic station noise level for probability of detection routine.

Called by `(show-best-channels)`.

`*prob-sd-noise*`

Specifies the characteristic station-noise standard-deviation level for probability of detection routine.

Called by `(show-best-channels)`.

`*prob-snr-thresh*`

Specifies the detection snr threshold for the probability of detection routine.

Called by `(show-best-channels)`.

`*prob-threshold*`

Specifies the probability threshold for `(show-best-chans)` to assume detection.

Called by `(show-best-channels)`.

`*regional-distance-cutoff* 20.0`

Specifies the maximum distance considered to be "regional."

Called by `(show-best-channels)`.

`sasc-directory-prefix`

Specifies the Slowness/Azimuth Station Corrections (SASC) file. A CVAR of a similar name `(sasc-dir-prefix)` exists in `ARSdefault.scm`.

Initial value:

```
(string-append (getenv "CMS_CONFIG") "/
earth_specs/SASC/sasc"))
```

Does not override standard.

`*scanning-waveform-height*`

Specifies the height to set for waveform windows when scanning.

Initial value: `45.0`

Calling object: `scan-region`

Overrides standard, but uses same value.

`sccs-id`

Specifies the source-specific station corrections (SCCS) identification for the *ARSdefault.scm* file. The *W* and *G* fields are filled automatically by SCCS when the file is checked out as "read-only."

`theoretical-color`

Specifies the color to paint theoretical arrivals. A CVAR with the same name exists in `ARSdefault.scm`.

Initial value: `"brown"`

Calling objects:
`paint-normal-objects`
`say-color-by-state-object`

Overrides standard.

`unassociated-color`

Specifies the color to paint unassociated arrivals.

`velocity-model-spec-file`

Specifies the location of the velocity-model-specification file (VMSF), which defines the travel-time tables. A CVAR of a similar name (`tt-velocity-model-spec-file`) exists in `ARSdefault.scm`.

Initial value:
```
(string-append (getenv "CMS_CONFIG") "/
earth_specs/TT/vmsf/ims.defs")
```

Calling objects:
`create-and-send-XfkDisplay-message`
`read-default-travel-time-tables`

Overrides standard.

`*waveform-start-fudge*`

Specifies a "fudge-factor" for existing-waveform checks.

Called by `(show-best-channels)`.

`*wc-max-smajax*`

Specifies the threshold for flagging large errors in origin location.

`wfdisc-directory`

Specifies the directory for reading/writing temporary *wfdisc* files used by *ARS* and *Beamer* or *SpectraPlot* (*ARS* is the reader; the other process is the writer). The user may redefine this directory to any valid directory.

LIST VARIABLES

`*alpha-list-arrivals*`

Stores the list of arrivals in the current AlphaList.

Initial value: `nil`

Calling objects:

```
show-alpha-list-and-remember-arrivals
find-alpha-list-arrivals
```

Does not override standard.

`*alpha-station-list*`

Specifies the list of alpha stations.

Initial value:

```
(list
;;
;; Seismic
;;
"ABKT" "ARCES" "ASAR" "BDFB" "BGCA" "BJT" "BOSA" "CMAR"
"CPUP" "DBIC" "ESDC" "FINES" "GERES" "HIA" "ILAR" "KBZ"
"KSAR" "LPAZ" "MAW" "MJAR" "MNV" "NOA" "NRIS" "PDAR" "PDY"
"PLCA" "ROSC" "SCHQ" "STKA" "TXAR" "ULM" "VNDA" "WRA" "YKA"
"ZAL"
;;
;;  Hydroacoustic
;;
"ASC19" "ASC21" "ASC26" "ASC27" "ASC29" "NZL01" "NZL06" "VIB"
"WK30" "WK31"
)
```

Calling objects:

```
check-for-minimum-alpha-stations
get-weighted-count
check-for-small-event-defining-rules
check-for-large-event-defining-rules
```

Overrides standard.

`arr-azdef-list`

Specifies a list of phases that default to be azimuth-defining at arrays.

`arr-slodef-list`

Specifies a list of phases that default to be slowness-defining at arrays.

`arr-timedef-list`

Specifies a list of phases that default to be time-defining at arrays.

`*array-identifiers*`

Specifies the *statype* strings delineating arrays (in the database).

Called by `(show-best-channels)`.

`*array-names*`

Specifies explicit array names that are distinct from individual element names.

Called by `(show-best-channels)`.

`*assoc-error-list*`

Temporary variable used for collecting error text.

Initial value: `nil`

Calling objects: `check-origin-list, assoc-error`

Does not override standard.

`azdef-list`

Specifies a list of phases that default to be azimuth-defining at non-array stations.

`*best-channels*`

Specifies the channels that will be displayed, whether or not `(show-best-chans)` selects them. The function `add-best-chans` will force display of these channels when `(show-best-chans)` is run.

Called by `(show-best-channels)`.

`*botf-qc-station-list* station-list`

Specifies a list of stations that will be checked during beam-on-the-fly processing.

`*botf-excluded-chans-list*`

Specifies a list of channels to be omitted during beam-on-the-fly processing.

`*channel-ordering*`

Specifies a list of channels used when sorting channels by their name.

Initial value:
```
'("cb" "ib" "hb" "bz" "bn" "be" "sz" "sn"
"se" "mz" "mn" "me" "lz" "ln" "le"))
```

Calling object: `get-channel-priority`

Does not override standard.

`chan-sort-list`

List that specifies the sort order for `(sort-distance-channel-channels)`, which may be bound to the Sort by Distance menu item.

`*default-component-list*`

Specifies the channels that `(show-best-chans)` will use if teleseismic or regional channels are not available.

Called by `(show-best-channels)`.

`*defining-phase-residuals-list*`

Compound list that specifies the allowable distance range and residuals for phases. Each component of the list is a list itself.

Initial value:

```
(list'
("Pg" 0.0 4.0 2.0 -1.0 5.0 2.0 -1.0 7.5 p-type-primary)
'("Pn" 0.0 4.0 2.2 -1.0 5.0 2.2 -1.0 7.5 p-type-primary)
'("Sn" 0.0 4.0 2.0 -1.0 5.0 2.0 -1.0 7.5 s-type-regional)
'("Lg" 0.0 4.0 3.0 -1.0 5.0 3.0 -1.0 7.5 s-type-regional)
'("Pg" 0.0 20.0 2.0 -1.0 7.5 2.0 -1.0 10.0 p-type-primary)
'("Pn" 0.0 20.0 2.2 -1.0 7.5 2.2 -1.0 10.0 p-type-primary)
'("Sn" 4.0 20.0 3.0 -1.0 7.5 3.0 -1.0 10.0 s-type-regional)
'("Lg" 4.0 20.0 4.5 -1.0 7.5 4.5 -1.0 0.0 s-type-regional)
'("P" 0.0 103.0 2.0 2.0 20.0 2.0 3.0 30.0 p-type-primary)
'("PKP" 110.0 180.0 2.5 2.0 30.0 2.5 3.0 45.0 p-type-primary)
'("PKPab" 143.0 180.0 2.5 2.0 30.0 2.5 -1.0 45.0 p-type-primary)
'("PKPbc" 145.0 155.0 2.5 2.0 30.0 2.5 -1.0 45.0 p-type-primary)
'("PKPdf" 110.0 180.0 2.5 2.0 30.0 2.5 -1.0 45.0 p-type-primary)
'("PKiKP" 20.0 180.0 2.5 2.0 30.0 2.5 -1.0 45.0 p-type-primary)
'("pP" 10.0 103.0 2.0 2.0 20.0 2.0 -1.0 30.0 p-type-secondary)
'("sP" 10.0 103.0 2.0 2.0 20.0 2.0 -1.0 30.0 p-type-secondary)
'("pPKP" 20.0 180.0 2.0 2.0 20.0 2.0 -1.0 30.0 p-type-secondary)
'("pPKPbc" 20.0 180.0 2.0 2.0 20.0 2.0 -1 0 30.0 p-type-second-
ary)
'("S" 0.0 103.0 6.0 2.0 20.0 6.0 -1.0 30.0 s-type-teleseismic)
'("PcP" 0.0 90.0 3.0 2.0 20.0 3.0 -1.0 -1.0 p-type-secondary)
'("ScP" 0.0 70.0 3.0 2.0 20.0 3.0 -1.0 -1.0 p-type-secondary)
'("SKP" 105.0 180.0 3.0 -1.0 -1.0 3.0 -1.0 -1.0 p-type-second-
ary)
'("SKPbc" 0.0 180.0 3.0 -1.0 -1.0 3.0 -1.0 -1.0 p-type-second-
ary)
'("PP" 20.0 180.0 3.0 3.0 20.0 3.0 -1.0 -1.0 p-type-secondary
))
```

Calling objects:

`*defining-phases*`

`*defining-phase-list*`

`get-weighted-contribution-from-assoc`

`find-phase-residual-info`

Overrides standard.

`*depth-phase-list*`

Specifies the list of phases that indicate an event at depth.

`*depth-sensitive-phase-list*`

Specifies a list of phases whose characteristics are particularly sensitive to the event depth.

`*geographic-scan-list*`

List of channel priorities used by `(scan-region)`. Stations are displayed in the order specified in this list.

Initial value:

```
(list
'("Australia / S. Hemisphere" (
("ASAR" ("fkb" "cb"))
("WRA"  ("fkb" "cb"))
("STKA" ("sz"))
("MAW"  ("bz"))
("VNDA" ("sz"))
("BDFB" ("sz"))
("PLCA" ("sz"))
("CPUP" ("sz"))
("LPAZ" ("bz"))
))
'("Europe / North Asia" (
("ARCES" ("fkb" "cb"))
("FINES" ("fkb" "cb"))
("NOA"   ("fkb" "cb"))
("GERES" ("fkb" "cb"))
("ESDC"  ("fkb" "cb"))
))
'("N. Africa / Asia / Mid East" (
("ABKT"  ("bz"))
("BGCA"  ("sz"))
("BOSA"  ("sz"))
("DBIC"  ("bz"))
("MJAR"  ("fkb" "cb"))
("KSAR"  ("fkb" "cb"))
("CMAR"  ("fkb" "cb"))
("PDY"   ("sz"))
("ZAL"   ("sz"))
))
'("W. Hemisphere / Americas" (
("YKA"   ("fkb" "cb"))
("TXAR"  ("fkb" "cb"))
("PDAR"  ("fkb" "cb"))
("ULM"   ("bz"))
("SCHQ"  ("bz"))
))
)
```

Calling objects:
`prompt-scan-region`
`scan-region`

Overrides standard.

`h-chans`

Stores a list of horizontal channels to be removed from the display by the function `unshow-horizontals`.

Initial value:

```
(list "be" "bel" "bn" "bnl" "ee" "en" "hb" "he" "hn" "ib"
"le" "ln" "me" "mel" "mn" "mnl" "se" "sn" "ue" "un" "ve"
"vn")
```

Calling object:
`unshow-horizontals`

Does not override standard.

`*hydro-stations*`

Specifies the list of hydroacoustic stations.

Initial value: `'("WK30" "WK31" "ASC19" "ASC21" "ASC26" "ASC27" "ASC29" "VIB" "NZL01" "NZL06")`

Calling objects:

```
show-hydro-chans
refsta-is-hydro-station
show-hydro-and-sort-chans
show-hydro-chans
select-hydro
lign-hydro-on-t
eck-and-retime-arrival
```

`*hydro-display-channels*`

Specifies which hydro channels to display. These channels are in the form to be used by `string->channels`. For example, for station PSUR to display channels PSUR0/sp and PSUR0/lp, the entry would be `("PSUR" "PSUR0/sp PSUR0/lp")`.

Initial value:

```
(list
'("WK30" "WK30/ed") '("WK31" "WK31/ed")
'("ASC19" "ASC19/ed")
'("ASC21" "ASC21/ed")
'("ASC26" "ASC26/ed")
'("ASC27" "ASC27/ed")
'("ASC29" "ASC29/ed")
'("VIB" "VIB/ez")
'("NZL01" "NZL01/ed")
'("NZL06" "NZL06/ed")
)
```

Calling object: `get-hydro-display-channel-string`

Overrides standard.

`list-of-arrival-remarks`

Specifies the list of remarks to use when prompting for an arrival remark.

Initial value:

```
(list
"Weak signal"
"Low confidence time, emergent"
"Low confidence phase ID"
"Low confidence association"
"Multiple, same az"
"Multiple, mixed az"
"Glitches"
"Dropouts"
"Noise bursts"
"High background noise"
"Known station timing error"
"Suspected station timing error")
```

`list-of-coda-phases`

Specifies a list of coda phase names.

Initial value: `(list "tx" "Px" "Sx")`

Calling objects:

`add-selectlist-associated-coda-arrivals,`
`add-selectlist-associated-other-arrivals`

Does not override standard.

`list-of-event-remarks`

Specifies a list of predefined event remarks.

Initial value:

```
(list
"Aftershock: abbreviated analysis"
"LO CONF"
"LO CONF Location"
"LO CONF Depth"
"Multiple, same az"
"Multiple, mixed az"
"Announced nuclear test")
```

Calling object: `prompt-remark-in-a-category`

Does not override standard.

`list-of-filters`

Specifies the list of filter values that may be applied to waveform data.

Initial value:

```
(list
"24.0 48.0 3 BP causal"
"12.0 24.0 3 BP causal"
"6.0 12.0 3 BP causal"
"6.0  9.9 3 BP causal"
"4.0  8.0 3 BP causal"
"3.0  6.0 3 BP causal"
"2.0  4.0 3 BP causal"
"1.5  3.0 3 BP causal"
"1.0  2.5 3 BP causal"
"1.0  2.0 3 BP causal"
"0.5  2.0 3 BP causal"
"0.4  1.5 3 BP causal"
"0.1  1.0 3 BP causal"
"0.01 0.1 3 BP causal"
"0.02 0.05 3 BP causal"
"0.001 10.0 1 BP causal"
"8.0 16.0 3 BP causal"
"3.0  5.0 3 BP causal"
"2.0  5.0 3 BP causal"
"1.0  5.0 3 BP causal"
"1.0  4.0 3 BP causal"
"1.0  3.0 3 BP causal"
"0.8  3.0 3 BP causal"
"0.7  2.0 3 BP causal"
"0.2  1.5 3 BP causal"
"0.03 0.1 3 BP causal"
)
```

Calling objects:

```
filter-prompt-selected-channels
add-multiple-filters
add-single-filter
edit-filter-dialog
modify-multiple-filters
prompt-create-cascade-filter
select-filter-dialog
```

Overrides the standard.

`list-of-fms`

Specifies the default list of first motion (*fm*) values for an arrival. These values are used in the prompt for arrival fm type. Each value is a quoted string, for example, `"cu"`.

`list-of-hydro-arrival-remarks`

Specifies a list of remarks that may be attached to an arrival; this list is used by `(prompt-remark-in-a-category)`.

Initial value:

```
(list
"T-phase, weak signal (low SNR)"
"T-phase, multiple energy peaks"
"T-phase, low confidence association"
"T-phase, low confidence time (emergent)"
"Impulsive (possible explosion)")
```

Calling objects: `prompt-remark-in-a-category`

Does not override standard.

`list-of-phases`

Specifies the default list of phases, for example, `"P"`. These phases are used in the prompt for arrival phase name in the `rename` and `create` functions.

Initial value:

```
(list
;Primary arrivals
"P" "Pn" "Pg" "PKP" "PKPbc" "PKPab"
"PKhKP" "PKiKP" "Pdiff"
;Secondary arrivals
"S" "Sn" "Lg" "Rg"
;Depth phases
"pP" "pPKP" "pPKPab" "pPKPbc" "sP"
 "sPKP"
;Depth sensitive phases
"PcP" "ScP"
;Other
"T" "H" "O" "I" "tx" "PP" "PPP" "LR" "LQ"
"PKKP" "PKKPbc"
"PKKPab" "PKKS" "P3KP" "P3KPbc" "P4KP"
"P4KPbc" "P5KP"
"P7KP" "PKP2" "PKP2ab" "PKP2bc" "SKP"
"SKPbc" "SKPab"
"SKiKP" "SKKP" "SKKPab" "SKKPbc"
"SKS" "SKKS" "ScS" "PcS" "PKS" "SS" "SP"
)
```

Calling objects:

```
prompt-phases-send-receive-beamer-messages
tt-phase
prompt-channel-theoreticals
prompt-phase-name set-chosen-phase
prompt-phase-and-align-channels
prompt-phase-create-arrival
prompt-and-rename-arrival
prompt-and-rename-arrivals
prompt-and-select-regional-theoreticals
prompt-and-select-teleseismic-theoreticals
prompt-phase-and-align-channels
prompt-phase-create-idcarrival
```

Overrides the standard.

`list-of-phases-with-travel-times`

Specifies the default list of phases used in the prompt for phase name for the following two functions that display the regional or teleseismic phases:

```
(prompt-and-select-regional-theoreticals)
(prompt-and-select-teleseismic-theoreticals)
```

This default list sets either `list-of-regional-theoreticals` or `list-of-teleseismic-theoreticals (phases)` to a new value.

Initial value:

```
(list "P" "Pb" "Pn" "Pg" "Sn" "Lg" "Rg"
"PKP" "pP" "sP" "PcP"
"PP" "PPP" "S" "SS" "SSS" "PS" "SP" "PPS" "PKKP" "ScS"
"PcS" "ScP" "T" "H" "O" "I" "SKS" "PKS"
"SKP" "LQ" "LR"
"PKPab" "PKKS" "PKPbc" "P3KP" "P3KPbc"
"P4KP" "P4KPbc"
"P5KP" "P7KP" "PKPPKP" "PKiKP" "pPKP"
"sPKP" "Pdiff" "pPKPab"
"pPKPbc" "pPKPdf" "pPKiKP" "PKP2"
"PKP2ab" "PKP2bc" "SKPab"
"SKPbc" "PKKPab" "PKKPbc" "SKKP"
"SKKPab" "SKKPbc"
"SKiKP" "SKKS")
```

Calling objects:
`prompt-phases-send-receive-beamer-messages`
(only in `ARSdefault.scm`; override in `IDC.scm`; does not include `list-of-phases-with-travel-times`)
`prompt-and-select-regional-theoreticals`
`prompt-and-select-teleseismic-theoreticals`
`prompt-phase-and-align-channels`
`read-default-travel-time-tables`

Overrides the standard.

`list-of-quals`

Specifies the default list of *qual* values, for, example, `"e"`. These values are used in `prompt-arrival-qual`.

`list-of-reasons`

Specifies a list for reasons for which an event may be discarded.

Initial value:

```
(list
"Invalid association"
"Invalid detections"
"Split event rejoined"
"Insufficient stations"
"Insufficient weight"
)
```

Calling objects:
`verify-orid-selected`
`set-selectlist-reasons!`

Does not override standard.

`list-of-regional-theoreticals`

Specifies the default list of theoretical arrivals, for example, `"Pn"`. These arrivals are used when `show-regional-theoreticals` is called. The default list may be set to a new value using `prompt-and-select-regional-theoreticals`.

`list-of-remarks`

Specifies the default list of *remark* values, for example, `"LA"`. These values are used in `prompt-object-remarks`.

`list-of-remark-categories`

Specifies a list of remark categories used by `(prompt-remark-by-category)`, which subsequently prompts the user for a remark.

Initial value:

```
(list
"Event,  Static remark"
"Event,  Arbitrary remark"
"Seismic Arrival, Static remark"
"Hydro Arrival, Static remark"
"Arrival, Arbitrary remark"
"Show Remarks")
```

Calling object: `prompt-remark-by-category`

Does not override standard.

`list-of-seismic-arrival-remarks`

Specifies a list of remarks that may be attached to an arrival; this list is used by `(prompt-remark-in-a-category)`.

Initial value:

```
(list
"Weak signal"
"Low confidence time, emergent"
"Low confidence phase ID"
"Low confidence association"
"Multiple, same az"
"Multiple, mixed az"
"Glitches"
"Dropouts"
"Noise bursts"
"High background noise"
"Known station timing error"
"Suspected station timing error")
```

Calling object: `prompt-remark-in-a-category`

Does not override standard.

`list-of-stassoc-etypes`

Specifies the default list of *stassoc etype* values, for example: `"de"`. These values are used in the prompt for `stassoc-etype`.

`list-of-stypes`

Specifies the default list of *stype* values, for example, `"d"`. These values are used in `prompt-arrival-stype`.

`list-of-teleseismic-theoreticals`

Specifies the default list of theoretical arrivals, for example, `"PKP"`. These arrivals are used when `prompt-and-select-teleseismic-theo-reticals` is called. The default list may be set to a new value using `prompt-and-select-teleseismic-theoreticals`.

`list-of-user-phases`

Specifies a list of phases that can be used for assigning phase names.

Initial value:
```
(list "P" "PKP" "Pn" "Pg" "Sn" "Lg" "Rg" "pP" "T" "sP" "S"
"PP" "PcP" "PKPbc" "PKPab" "Pdiff" "PKiKP" "PKKP" "PKPPKP")
```

Calling objects:
`add-selectlist-user-associated-arrivals`
`add-selectlist-associated-other-arrivals`

Does not override standard.

`measurement-amptypes-list`

Specifies the list of available *amptypes* that can be measured.

`*non-defining-list*`

Specifies a temporary list of phases that have been flagged as non-defining due to errors.

Initial value: `nil`

Calling objects:
`check-origin-list`
`check-defining`
`set-non-defining`

Does not override standard.

`not-assoc-list`

Specifies the list of phases to exclude from automatic association.

`*origin-error-list*`

Temporary variable used for collecting error text.

Initial value: `nil`

Calling objects:
`check-origin-list`
`add-origin-error`
`add-assoc-error`

Does not override standard.

`*p-type-phases*`

Specifies a list of P phases used by `(check-origin-for-two-p-phases)`.

Initial value: `'("P" "Pn" "PKP" "Pg")`

Does not override standard.

`*phase-azres-table*`

Specifies a list of phases and their acceptable azimuth residuals; used by `(check-assoc-azimuth-residual)`.

Initial value:

```
'(("Pg" () 4 5.0)
("Pn" () 4 5.0)
("Px" () 4 5.0)
("Sn" () 4 5.0)
("Px" () 4 5.0)
("Lg" () 4 5.0)

("Pg" 4 () 7.5)
("Pn" 4 () 7.5)
("Px" 4 () 7.5)
("Sn" 4 () 7.5)
("Sx" 4 () 7.5)
("Lg" 4 () 7.5)

("P" () () 20.0)
("S" () () ())
("pP" () () ())
("sP" () () ())
("PcP" () () ())
("PKP" () () 30.0)
("PKPab" () () ())
("PKKP" () () ())
("PKP2" () () ())
("PKP2ab" () () ())
("PKP2bc" () () ())
("PKiKP" () () ())
("SKP" () () ())
("PP" () () ())
("tx" () () 20))
```

Calling objects: `check-assoc-azimuth-residual`

Does not override standard.

`*phase-slores-table*`

Specifies a list of phases and their acceptable slowness residuals; used by `(check-assoc-slowness-residual)`.

Initial value:

```
'(("Pg" () 4 ())
("Pn" () 4 ())
("Px" () 4 ())
("Sn" () 4 ())
("Sx" () 4 ())
("Lg" () 4 ())

("Pg" 4 () ())
("Pn" 4 () ())
("Px" 4 () ())
("Sn" 4 () ())
("Sx" 4 () ())
("Lg" 4 () ())

("P" () () 2.0)
("S" () () ())
("pP" () () ())
("sP" () () ())
("PcP" () () ())
("PKP" () () 3.0)
("PKPab" () () ())
("PKKP" () () ())
("PKP2" () () ())
("PKP2ab" () () ())
("PKP2bc" () () ())
("PKiKP" () () ())
("SKP" () () ())
("PP" () () ())
("tx" () () ()))
```

Calling object: `check-assoc-slowness-residual`

Does not override standard.

`*phase-timeres-table*`

Specifies a list of phases and their acceptable time residuals; used by `(check-assoc-time-residual)`.

Initial value:

```
'(("Pg" () 4 1.75)
("Pn" () 4 2.25)
("Px" () 4 ())
("Sn" () 4 2.0)
("Sx" () 4 ())
("Lg" () 4 3.0)
("Pg" 4 () 1.75)
("Pn" 4 () 2.25)
("Px" 4 () ())
("Sn" 4 () 2.0)
("Sx" 4 () ())
("Lg" 4 () 3.0)
("P" () () 2.0)
("S" () () 6.0)
("pP" () () 1.5)
("sP" () () 1.5)
("PcP" () () 3.0)
("PKP" () () 2.5)
("PKPab" () () 2.5)
("PKKP" () () 4.5)
("PKP2" () () 4.5)
("PKP2ab" () () 4.5)
("PKP2bc" () () 4.5)
("PKiKP" () () 4.5)
("SKP" () () 4.5)
("PP" () () 3.0)
("tx" () () ()))
```

Calling object: `check-assoc-time-residual`

Does not override standard.

`*primary-phase-list*`

Specifies a list of phases considered to be primary.

`*pseudo-origin-locations*`

Specifies the list of locations and descriptions for the analyst special location set. This list is used in the function `prompt-create-pseudo-origin`.

Called by `(show-best-channels)`.

Initial value:

```
'(" 11.0  -88.0 30.0  Central America"
"  38.0   22.0 30.0  Mediteranian"
"  12.0  125.0 30.0  Philippines"
"  44.0  145.0 30.0  Kuriles/Kamchatka"
"  51.0 -172.0 30.0  Andreanof Islands"
" -24.0 -178.0 30.0  Fiji Islands"
" -56.0  -26.0 30.0  Sandwich Islands"
"  -7.0  126.0 30.0  Banda Sea"
" -23.0  -69.0 30.0  Andes (S. America)"
"  36.0   69.0 30.0  Hindu Kush"
)
```

Calling objects:

```
prompt-create-pseudo-origin
prompt-delete-pseudo-origin-location
add-pseudo-origins-from-selected-origins
```

Overrides standard.

`*regional-array-components-for-display*`

Specifies channels to display at arrays, if the station/event distance is regional.

Called by `(show-best-channels)`.

`*regional-array-filter-parameters*`

Specifies the filter parameters for regional-distance channels (arrays).

Called by `(show-best-channels)`.

`*regional-phase-list*`

Specifies the phases that are generally only seen for regional events.

`*regional-ss-filter-parameters*`

Specifies the filter parameters for regional-distance stations.

Called by `(show-best-channels)`.

`*reasonable-magnitude-limits*`

Specifies a list with the minimum and maximum magnitudes that are likely to result from IDC measurements. Values outside this range may trigger warnings.

`*regional-ss-components-for-display*`

Specifies a list of short-period components to display if an event is at regional distance.

Initial value: `'("bz" "bn" "be" "sz" "sn" "se")`

Calling object: `get-proper-component-for-station`

Overrides standard, but uses same value.

`*station-azimuth-slowness-reliability-list*`

Specifies a list of the empirical reliability of the slowness measures from various stations. The format for each element is a list in the following form: (*Station Azimuth-status Slowness-status*).

Initial value:

```
(list
'("ABKT"    unreliable     unknown)
'("ARCES"   reliable       unknown)
'("ASAR"    reliable       unknown)
'("BDFB"    moderate       unknown)
'("BGCA"    reliable       unknown)
'("BJT"     unknown        unknown)
'("BOSA"    moderate       unknown)
'("CMAR"    reliable       unknown)
'("CPUP"    unreliable     unknown)
'("DBIC"    moderate       unknown)
'("ESDC"    reliable       unknown)
'("FINES"   reliable       unknown)
'("GERES"   reliable       unknown)
'("HFS"     moderate       unknown)
'("HIA"     unknown        unknown)
'("KBZ"     unreliable     unknown)
'("LBNH"    unreliable     unknown)
'("LOR"     unreliable     unknown)
'("LPAZ"    moderate       unknown)
'("MAW"     moderate       unknown)
'("MBC"     moderate       unknown)
'("MIAR"    unreliable     unknown)
'("MJAR"    moderate       unknown)
'("NORES"   reliable       unknown)
'("NPO"     unreliable     unknown)
'("NRIS"    unreliable     unknown)
'("PDAR"    unreliable     unknown)
'("PDY"     unreliable     unknown)
'("PFO"     unreliable     unknown)
'("PLCA"    moderate       unknown)
'("SCHQ"    unreliable     unknown)
'("SPITS"   unreliable      unknown)
'("STKA"    reliable        unknown)
'("TXAR"    reliable        unknown)
'("ULM"     moderate        unknown)
'("VNDA"    moderate        unknown)
'("WALA"    unreliable      unknown)
'("WHY"     unreliable      unknown)
'("WOOL"    moderate        unknown)
'("WRA"     reliable        unknown)
'("YKA"     reliable        unknown)
'("ZAL"     unreliable      unknown)
)
```

Calling objects:
`find-station-azimuth-reliability`
`find-station-slowness-reliability`

Overrides standard.

`slodef-list`

Specifies a list of phases that default to be slowness-defining at non-array stations.

`*station-filter-overrides*`

Specifies filter selection for selected stations. These filters will override the filters in the variables `*regional-array-filter-parameters*` and so on.

Called by `(show-best-channels)`.

`*superset-list*`

Specifies a set of stations to always be included in the display list.

Called by `(show-best-channels)`.

`*station-component-overrides*`

Specifies component selection for selected stations. These channels will override the components in the following variables: `*regional-array-components-for-display*`, `*teleseismic-array-components-for-display*`, and so on.

Called by `(show-best-channels)`.

Initial value:

```
(list
'("ABKT" ("bz"))
'("ARCES" ("cb" "fkb"))
'("ARA0" ("sz"))
'("ARE0" ("iz"))
'("ASAR" ("cb" "fkb"))
'("AS12" ("sz"))
'("ASPA" ("sz"))
'("BDFB" ("bz"))
'("BJT" ("bz"))
'("BGCA" ("bz"))
'("BOSA" ("bz"))
'("CMAR" ("fkb" "cb"))
'("CM06" ("sz"))
'("CM31" ("bz"))
'("CPUP" ("bz"))
'("DBIC" ("bz"))
'("ESDC" ("cb" "fkb"))
'("ESLA" ("bz"))
'(FINES" ("fkb" "cb"))
'("FIA0" ("sz"))
'("GERES" ("cb" "fkb"))
'("GEC2" ("hz"))
'("HFS" ("cb" "fkb"))
'("HFSC2" ("sz"))
'("HIA" ("bz"))
'("KBZ" ("sz"))
'("LPAZ" ("bz"))
'("MAW" ("bz"))
'("MBC" ("bz"))
'("MJAR" ("cb" "fkb"))
'("MJ00" ("ez"))
'("NRA0" ("sz"))
'("NRE0" ("iz"))
'("NRIS" ("sz"))
'("NPO" ("sz"))
'("PDAR" ("cb" "fkb"))
'("PD05" ("sz"))
'("PDY" ("sz"))
'("PLCA" ("bz"))
'("SCHQ" ("bz"))
'("STKA" ("bz"))
'("TXAR" ("cb" "fkb"))
'("TX00" ("bz"))
'("ULM" ("bz"))
'("VNDA" ("bz"))
'("WRA" ("cb" "fkb"))
'("WR1" ("sz"))
'("WR5" ("sz"))
'("YKA" ("cb" "fkb"))
'("YKR8" ("sz"))
'("YKW3" ("bz"))
'("ZAL" ("sz"))
)
```

Calling object: `get-override-chans`

Overrides standard.

`*teleseismic-array-components-for-display*`

Specifies channels to display at arrays if the station/event distance is teleseismic.

Called by `(show-best-channels)`.

`*teleseismic-array-filter-parameters*`

Specifies filter parameters for teleseismic-distance channels at arrays.

Called by `(show-best-channels)`.

`*teleseismic-ss-components-for-display* '("bz"))`

Specifies channels to display at stations that are not arrays, if the station-event distance is teleseismic.

Called by `(show-best-channels)`.

`*teleseismic-ss-filter-parameters*`

Specifies filter parameters for teleseismic-distance stations that are not arrays.

Called by `(show-best-channels)`.

`timedef-list`

Specifies a list of phases that default to be time defining at non-array stations.

`*unfilterable-components*`

Specifies a list of components that cannot be filtered.

Called by `(show-best-channels)`.

`*wc-array-list*`

Specifies the list of array stations.

`*wc-hydro-list*`

Specifies the list of hydroacoustic stations.

`*wc-origin-uncertainty-remarks*`

Specifies the list of origin remarks that identify an origin as having a low location confidence.

C VARIABLES

This section lists the *C* variables (CVARs), which are also used directly by the *ARS* compiled code and are set or retreived using the functions shown in Table 1.

TABLE 1: CVAR SUPPORT FUNCTIONS

Function	Description
`(set-cvar! "cvar-name" value)`	create/set value
`(get-cvar-string "cvar-name")`	retrieve string value
`(get-cvar-number "cvar-name")`	retrieve numeric value

The variable names are shown along with their default values.

`alphalist-chan-sta-background-color: AliceBlue`

Specifies the background color for channels in the alpha list.

`alphalist-defining-background-color: steelblue`

Specifies the background color of defining stations in the alpha list.

`alphalist-detect-phase-background-color: AliceBlue`

Specifies the background color of defining detections in the alpha list.

`alphalist-generic-background-color: lightsteelblue`

Specifies the default background color of cells in the alpha list.

`alphalist-orig-orid-background-color: AliceBlue`

Specifies the default background color of origin cells in the alpha list.

`amplitude-units: nanometers`

Specifies the units for writing amplitude, which are either digital counts or nanometers (default). Nanometers are counts corrected by the scalar correction given by *calib* in wfdisc. Valid arguments are `counts` or `nanometers`.

`ars-auto-write-function-list`

Specifies a space-delimited string of *Scheme* functions that should call the auto write function associated with *ARS* crash recovery.

`broken-associated-origin-color: orange`

Specifies the color to paint arrivals that are associated with broken origins.

`broken-color: red`

Specifies the color to paint broken objects.

`channel-color: black`

Specifies the color to paint channel objects.

`check-LR-amp-out-of-range-p: False`

Checks if the period for LR amplitudes is in the 18 to 24-second range. This check occurs after the amplitude and period are measured; if this predicate is `True` and the period is invalid, the measurements are not accepted, and an error box is displayed.

`color-code-p: True`

Uses the value returned by `(say-color-by-state-object` *obj*`)` to determine whether or not *ARS* objects are displayed by color.

`color-in-scheme-p: False`

Determines whether to (`True`) use the *Scheme* function (`paint-code-objects`) and the *Scheme* variables to determine coloring, or (`False`) use color defined in *C* and CVARs. For example, `channel-color` has two definitions for the two methods of coloring:

```
(define channel-color "black")
(set-cvar! "channel-color" channel-color)
```

`constant-amplitude-period-display-p: True`

Determines if the amplitude-period box used for measuring amplitude and period should remain displayed after the measurement is completed. When evaluated as `True`, the box remains until the display is redrawn.

`current-color: blue`

Specifies the color to paint current completed objects.

`database: none`

Specifies the database that *ARS* uses for all parameters.

`database-vendor: oracle`

Specifies the database vendor; currently, only ORACLE is supported.

`default-channels: none`

Specifies the channels to display on the initial screen.

`default-color: black`

Specifies the default color with which to draw objects.

`derived-channel-color: grey 10`

Specifies the color to paint derived channel objects.

`det-auto-deltim-p: True`

Specifies whether or not *deltim* should be automatically computed from snr for associated arrivals when a location is computed.

`det-default-snr: 7.9`

Specifies the default snr to use for arrivals without computed snr's such as analyst-added ones.

`det-max-deltim: 1.07`

Specifies the allowable maximum *deltim* when computed from snr.

`det-min-deltim: 0.12`

Specifies the allowable minimum *deltim* when computed from snr.

`det-max-snr: 17.7`

Specifies the maximum snr to use when computing *deltim* from snr.

`det-min-snr: 3.0`

Specifies the minimum snr to use when computing *deltim* from snr.

`dim-offchan-arrivals-p: False`

Allows an arrival to be dimmed (drawn dotted) if displayed on a channel on which it was not detected.

`duration: 86400`

Specifies the duration of the database window to read.

`filter-parameters: "1.0 3.0 3 BP 0"`

Specifies the low-cut frequency, high-cut frequency, number of poles, type of filter, and causality of a filter to apply.

`frozen-color: green`

Specifies the color of frozen objects.

`hydro-phase-list`

Specifies the space-delimited string of hydroacoustic phases.

`list-of-beams: cbP cb hb zb`

Specifies the channel list by which beams are recognized. These beams are displayed/undisplayed according to tags in the **wftag** table.

`locator-alpha-list-sort-func: (False)`

Specifies whether or not to resort the alpha-numeric list after event relocation. When set to `(False)`, the variable prevents resorting; when set to `(show-alpha-list)`, it creates the alpha-numeric list after event relocation, but does not actually display the list.

`locator-bypass-gui-p: False`

Determines whether or not the locator GUI should be displayed. If `True`, the locator and magnitude windows will not be shown.

`locator-chan-sort-func: (False)`

Controls sorting of waveforms after event relocation. If set to `(False)`, waveforms are not sorted after event relocation. If set to `(qsort-distance-channels)`, waveforms are resorted.

`locator-confidence-level: 0.9`

Specifies the confidence probability level for computing error ellipse information. Only `0.90`, `0.95`, and 0.99 (90th, 95th and 99th percentile) levels are accepted as valid values.

`locator-damping-factor: -1.0`

Specifies the applied system (sensitivity matrix) damping as a percentage of its largest singular value. The locator will determine its own best internal damping value when this factor is set to a negative value.

`locator-degrees-freedom: 99999`

Specifies the former number of degrees of freedom as defined by [Jor81]. This parameter, coupled with the former variance scale factor, `locator-sigma0`, incorporates past event knowledge into the current data set. The result is a more highly representative balance between the newly defined data residuals and the data standard errors. Therefore, the result will incorporate both the former and latter information into the final error ellipsoid calculations. Any positive value, including 0 is acceptable. The resultant error ellipse calculated with 0 degrees of freedom is equivalent to the traditional confidence ellipse of [Fli65], which assumes that only the final data residuals dictate the size of the error ellipse. However, an error ellipse calculated with infinite degrees of freedom is equivalent to the coverage ellipse of [Eve69] and assumes the former information is known exactly. In practice, just a very large number is used to represent this latter (chi-squared) case.

`locator-depth: 0.0`

Specifies the initial hypocentral depth (km). If depth is constrained (fixed), this value will remain constant throughout the iterative location procedure.

`locator-dist-var-wgt-p: True`

Applies distance variance weights to arrival time data, if value is `True`.

`locator-ellip-cor-type: 2`

Specifies the type of ellipticity correction to apply (none[`0`], AFTAC[`1`], and [Dzi75] [`2`]). For IMS applications, always use `2`.

`locator-fixed-depth-p: True`

Constrains (fixes) the depth (`True`[1] and `False`[0]).

`locator-fixed-latlon: 0`

Specifies which fixed latitude/longitude to use.

`locator-fixed-latlon-p: False`

Constrains (fixes) the origin time (`True`[1] and `False`[0]).

`locator-fixed-origin-time: 0;;;`

Specifies the fixed origin time to use.

`locator-fixed-origin-time-p: False`

Constrains (fixes) the origin time (`True`[1] and `False`[0]).

`locator-ignore-large-res-p: False`

If `True`, ignores (does not use) travel-time data where its residual (sec.) is greater than `locator-large-res-mult` multiplied by its variance weight.

`locator-large-res-mult: 3.0`

If `True`, uses only arrival data contained in the substation list `locator-sub-stations-list`. This `set-cvar!` can be used in other ways as well to restrict which stations are used to determine magnitude.

`locator-latitude: -90.0`

Specifies the initial latitude position (deg). If set within a valid range (–90.0 <- lat <- 90.0), this value will be used to override the latitude set in the origin table.

`locator-longitude: -180.0`

Specifies the initial longitude position (deg). If set within a valid range (–180.0 <- lon <- 180.0), this value will be used to override the longitude set in the **origin** table.

`locator-max-iterations: 60`

Specifies the maximum number of iterations allowed in determining an event location.

`locator-origin-time: 0.0`

Specifies the initial origin time (epoch sec). If set within a valid range (not `0.0`), this value will be used to override the origin time set in the **origin** table.

`locator-sigma0: 1.0`

Specifies an estimate of the data standard errors for a given event (specifies a former estimate for the variance scale factor). The variance scale factor can be interpreted as the mean ratio between the actual and assumed data variances. This variable determines sigma0 from the normalized sample variances computed during a series of previously determined event locations. Sigma0 is closely related to `locator-degrees-freedom` in that if former variance is known exactly, the number of degrees of freedom are infinite; then sigma0 must be 1.0, which implies that errors are independent of the data residuals and depend only on the model covariance matrix (the geometry of the network and the assigned variances). If the number if degrees of freedom are less than infinity, sigma0 must be based on the standard deviations of past events. If sigma0 is greater than 1.0, the given data variances underestimate the true variance of the input arrival data. The opposite is true if sigma0 is less than 1.0.

`locator-srst-var-wgt-p: False`

Applies Source Region Station Time (SRST) travel-time corrections if the event is contained within a valid SRST region. (`TRUE/FALSE`)

`locator-sssc-level: 0`

Specifies level of Source Specific Station Correction (SSSC) to be applied when determining travel time, azimuth, or amplitude (none[`0`], regional[`1`], and local[`2`]). Regional and local SSSC will only be applied if they exist for a given station/phase/region.

`locator-sub-stations-list: 12 BC03 BM05 CM16 IL01 IM03 KS15 SOBB TT01`

Used with `locator-sub-stations-only-p`, this list restricts which stations are used to determine an event location.

`locator-sub-stations-only-p: False`

If `TRUE`, uses only arrival data contained in substation list, `locator-sub-stations-list`. In *ARS* this is controlled by the Locator Control GUI toggle button, Sub-Stations Only. This `set-cvar!` can be used in other ways as well to restrict which stations ultimately get used in determining magnitude.

`locator-test-site-region:`

If `locator-use-test-site-corr-p` is `True`, then this variable applies test-site corrections at this region. This correction, if it exists, will supersede Source Specific Station Correction (SSSC) and Source Region Station Time (SRST) corrections if they are erroneously requested.

`locator-test-site-region-pairs:`

Specifies a list of test-site region pairs; each test-site region is a paired CVAR `locator-test-site-region` set to this designator, for example, "Lop Nur, LN, Degelen Mountain, DM," for the two pairs "Lop Nur, LN" and "Degelen Mountain, DM." The syntax is "*test-site-region*, *2-letter-designator*" with a comma separating additional pairs.

`locator-triple-location-p: True`

Specifies whether or not three location hypotheses (surface, free, and restrained) should be computed for each event.

`locator-use-current-location-p: True`

Specifies whether or not to use the current latitude and longitude components of the given event location. If `True`, current latitude and longitude are used as the starting event location. If `False`, the locator will determine its own best-guess starting location. This variable is usually set to `False` when the user is so far away from the desired event minima that convergence is difficult or impossible. As a general rule, reset this `set-cvar` to `True` after you are finished with it.

`locator-use-elev-corr-p: True`

Applies elevation corrections (`TRUE`[1] and `FALSE`[0]). This variable should always be set to `TRUE`, except for consistency testing of old versions of *ARS*.

`locator-use-only-sta-w-corr-p: False`

Specifies to only use data with valid Source Specific Station Correction (SSSC) and Source Region Station Time (SRST) or test-site corrections to construct the event. This variable ignores lower-quality information (in a modeling sense) by eliminating datums with assumed larger modeling errors.

`locator-use-srst-p: False`

Applies Source Region Station Time (SRST) variance weighting if the event is contained within a valid SRST region. (`TRUE`/`FALSE`)

`locator-use-test-site-corr-p: False`

See `locator-test-site-region`.

`locator-user-var-wgt: -1.0`

Specifies the variance weight applied when `locator-user-var-wgt-p` is true.

`locator-user-var-wgt-p: False`

Applies a former user variance weight to all arrival time data. This variable is subjective and not recommended. If `TRUE`, a user-variance weight equivalent to `locator-user-var-wgt` is applied to all data regardless of its quality.

`locator-verbose-file: ""`

Specifies the file to which all output is written, if the value is not `null`. If `null`, all output is written to the screen. The level of verbosity is set in `locator-verbose-p`.

`locator-verbose-p: 2`

Specifies the level of verbosity for printed locator output, scaled from `0` (no printed output) to `4` (all output printed). Intermediate values of `1`, `2`, or `3` bound these two extremes with detail increasing with the desired level. (The old settings of `y` and `n` still function as before, however, now `y = 4` and `n = 0`.)

`magnitude-always-magdef: False`

Specifies if station magnitudes should always be magnitude defining.

`magnitude-arr-amptype-const-p: False`

Predicate that determines if measured *amptypes* should always be a constant value of `magnitude-arr-amptype-const-value` (often A5/2). If this predicate is `True`, the *amptype* will always be this value.

`magnitude-arr-amptype-const-value`

Specifies the *amptype* to use if `magnitude-arr-amptype-const-p` is true.

`magnitude-ignore-large-res-p: False`

Specifies whether or not to ignore (not use) travel-time data where its residual (sec.) is > `locator-large-res-mult` multiplied by its variance weight. If `True`, the data are ignored.

`magnitude-large-res-mult: 3.0`

Applies large residual multiplier when `magnitude-ignore-large-res-p` is `True`. Usually this variable is set to 3.0.

`magnitude-magtype-list: mb mb_mle ms ms_mle`

Specifies the list of acceptable *magtypes*.

`magnitude-magtype-to-origin-mb: mb`

Specifies which *magtype* to write to the origin tuple as m_b.

`magnitude-magtype-to-origin-ml: ml`

Specifies which *magtype* to write to the origin tuple as M_L.

`magnitude-magtype-to-origin-ms: ms`

Specifies which *magtype* to write to the origin tuple as M_s.

`magnitude-max-mb-dist: 100`

Specifies the maximum distance in degrees at which a station contributes amplitude information to the m_b magnitude calculation. If `magnitude-use-sta-gt-max-dist` is `False`, then a station *delta* must be less than this distance (and possibly greater than `magnitude-max-mb-dist`, too).

`magnitude-mb-avg-magtype: mb`

Specifies the *magtype* used for network m_b magnitude; this *magtype* is used for reference in `magnitude-mdf-filename` and `magnitude-tl-model-filename`.

`magnitude-mb-magtype-list: mb mb_mle`

Specifies the list of acceptable *magtypes* for m_b.

`magnitude-mb-mle-magtype: mb_mle`

Specifies the *magtype* used for maximum-likelihood m_b magnitude.

`magnitude-mb-phases: P PKPdf NP`

Specifies the phases to use for m_b computation.

```
magnitude-mdf-filename: (string-append (getenv “CMS_CONFIG”)
“/ earth_specs/MAG/mdf/idc_mdf.defs”)
```

Specifies the location of the Magnitude Description File (`mdf`), which is the file that specifies the high-level magnitude and the computational control settings. This variable also describes the mappings to the Transmission Loss Specification File (TLSF).

`magnitude-min-mb-dist: 20`

Specifies the minimum distance in degrees at which a station contributes amplitude information to the m_b magnitude calculation. If `magnitude-use-sta-lt-min-dist` is `False`, then a station delta must be greater than this distance (and possibly less than `magnitude-max-mb-dist`, too).

`magnitude-ms-avg-magtype: ms`

Specifies the *magtype* used for network M_s magnitude.

`magnitude-ms-mle-magtype: ms_mle`

Specifies the *magtype* used for maximum-likelihood M_s magnitude.

`magnitude-nboot: 20`

Specifies the number of iterations of data resampling for bootstrap estimation (*nboot* = 0; no bootstrap resampling). If non-zero, *nboot* randomly selected data sets will be constructed, computed, and averaged to obtain better error estimates less likely affected by data biases. If bootstrapping, *nboot* should be at least `10`; `20` is recommended.

`magnitude-prt-verbose: 0`

Specifies the print setting. `0` = no output, `1` = network magnitude information only, `2` = network and station magnitude information.

```
magnitude-sub-stations-list: ABKT ARCES ASAR BDFB BGCA BJT BOSA CMAR
CPUP DBIC ESDC FINES GERES HIA ILAR KBZ KSAR LPAZ MAW MJAR MNV NOA NRIS
PDAR PDY PLCA ROSC SCHQ STKA TXAR ULM VNDA WRA YKA ZAL
```

When used with `magnitude-sub-stations-only-p`, this list restricts which stations are used to determine the network magnitude.

`magnitude-sub-stations-network: IDC`

Specifies the name of the network for a substation list. The value is populated in the *net* field of **database table**, and *netmag*, when the above `set-cvar! magnitude-sub-stations-only-p` is set to `True`.

`magnitude-sub-stations-only-p: True`

Called by `save-and-run-EvLoc`, this variable, when `True`, uses only magnitude data contained in the substation list, `magnitude-sub-stations-list`, to restrict which stations ultimately get used in determining magnitude. In *ARS* this setting is controlled by the Magnitude Control GUI toggle button, "Sub Stations Only." This `set-cvar!` can be used in other ways as well.

`magnitude-test-site-region:?`

Specifies the magnitude test-site region requested. Also see the descriptions of `magnitude-test-site-region` and `magnitude-test-site-region-pairs` and those variables that define the *magtype* of m_b and M_s magnitudes with test-site corrections.

`magnitude-test-site-region-pairs: none`

Specifies a list of test-site region pairs. Each test-site region is paired with a two-letter designator, which is used to set the CVAR `magnitude-test-site-region`. An example would be "Lop Nur, LN, Degelen Mountain, DM," for the two pairs "Lop Nur, LN" and "Degelen Mountain, DM". The syntax is "*test-site-region*, *2-letter-designator*" with a comma separating additional pairs.

`magnitude-tl-model-filename: (string-append (getenv "CMS_CONFIG") "/earth_specs/MAG/tlsf/idc_tlsf.defs")`

Specifies the location of the Transmission Loss Specification File (TLSF), which defines all regionalized transmission loss (TL) knowledge.

`magnitude-use-only-sta-w-corr-p: False`

Specifies whether or not to use only amplitude data with a valid test-site correction available on subsequent calls. If `True`, this variable sets magnitude defining switches to `False` for any data that do not have a corresponding test-site correction. This setting is meant to only affect subsequent calls to this routine, not the current one.

`magnitude-use-sta-gt-max-dist: False`

> Specifies whether or not to use station data with distances greater than `magnitude-max-mb-dist` in the magnitude calculation. If `True`, this variable uses amplitude information for stations greater than `magnitude-max-mb-dist` from the event. (Default: `False`)

`magnitude-use-sta-lt-min-dist: False`

> Specifies whether or not to use station data with distances less than `magnitude-min-mb-dist` in magnitude calculation. If `True`, this variable uses amplitude information for stations less than `magnitude-min-mb-dist` from the event. (Default: `False`.)

`magnitude-use-test-site-corr-p: False`

> Specifies whether or not (`True`/`False`) to apply magnitude test-site corrections.

`magnitude-weight-p: False`

> Specifies whether or not variable weighting should be applied to phase data in the process of determining a magnitude.

`max-wfdisc-duration: 86400`

> Specifies the maximum expect length, in seconds, of a *wfdisc* file. This variable is used when searching for *wfdisc*'s that include a specific time.

`measurement-amptype: A5/2`

> Specifies the default amplitude type when an amplitude is measured.

`network: PRI`

> Specifies which default network to use.

`not-current-associated-arrival-color: purple`

> Specifies the color to use for arrivals that are not associated with a not-current origin.

`not-current-associated-origin-color: purple`

Specifies the color to paint arrivals that are associated with origins that are not current: Their location hypothesis is not current with associated data.

`perm-wfdisc-directory: none`

Specifies the directory to which newly created beams are written.

`perm-wfdisc-extra-directory: none`

Specifies an extra directory name that is needed for beam waveform files.

`phases: LQ LR Lg P PKP PP PcP Pg Pn Px Rg S SKS SS ScS Sn Sx pP sP N T`

Specifies the default list of phases used in the prompt for arrival phase name in the `rename` and `create` functions.

`sasc-dir-prefix: /vobs/rel/pidc_6.0/rel/ops/static/SASC/idc_sasc`

Specifies the location of Slowness/Azimuth Station Correction (SASC) directory pathway and prefix name. The default structure should resemble *OPS_DIR_TREE*`/ops/static/SASC/`*PREFIX_NAME* where *OPS_DIR_TREE* defines the operational directory (for example, `/prj/shared`), and *PREFIX_NAME* specifies the actual prefix pre-pended to each SASC filename (for example, `sasc`).

`show-detect-bar-p: True`

Predicate that specifies whether or not the vertical bars for arrivals should be shown.

`show-detect-label-p: False`

Predicate that specifies whether or not the phase labels for arrivals should be shown.

`show-filter-parameters-p: True`

Predicate that specifies whether or not the filter parameters for filtered channels should be shown.

`show-origins-with-no-assocs-p: True`

Predicate that specifies whether or not origins without associated arrivals should be shown.

`show-scale-type-p: False`

Specifies whether or not the type of applied scaling should be shown.

`show-waveforms-p: False`

Specifies whether or not waveforms should be shown.

`snap-p: False`

Specifies whether or not amplitude measurements should "snap" to local minima and maxima.

`sort-alpha-arr-by-dist: True`

Specifies whether or not arrivals in the alpha list should be sorted by the distance between its detecting station and its associated origin's hypocenter. This variable is used only if `sort-in-scheme-p` is `false`, that is, if sorting is done in *C*.

`sort-in-scheme-p: False`

Called by `qsort-alpha-list`, this variable specifies whether or not to sort the alpha list in *Scheme* or instead, use the built-in *C* sorting functions. If this variable is set to `False`, no changes are necessary in any *Scheme* functions.

`start-time: none`

Specifies the start time of the time window for which to read data.

`theoretical-color: yellow`

Specifies the color to be used for theoretical arrivals.

`tt-velocity-model-spec-file:`
`/vobs/rel/pidc_6.0/rel/ops/static/TT/vmsf/ims.defs`

Specifies the location of the velocity model specification file (VMSF), which defines the travel-time tables. The format is *directory_pathway/filename*. The default structure should resemble

OPS_DIR_TREE`/ops/static/TT/vmsf/`*FILENAME*

where *OPS_DIR_TREE* defines the operational directory (for example, `/prj/shared`), and *FILENAME* specifies the actual filename of the desired VMSF.

`unassociated-color: red`

Specifies the color to paint arrivals that are not associated with any origin.

`use-peak-time-for-hydro-arrivals: False`

Predicate that specifies whether or not the maximum amplitude should be used for the timing of hydro arrivals.

`user-name: none`

Specifies the name of the user.

`vary-detect-height-p: False`

Predicate that specifies whether or not the size of the arrival bars should be scaled to indicate the snr.

`wfdisc-par-file: none`

Specifies the par file to use if `perm-wfdisc-directory` is not set.

ARS-specific Scheme Functions

This chapter provides an alphabetical listing and description of all *ARS*-specific *Scheme* functions.

ARS-specific Scheme Functions

This section describes the functions defined specifically to support the *ARS* application. The location of each function is identified as one of three sources: `ARSdefault.scm`, `IDC.scm`, or a *C* function linked into the *ARS* binary program. `IDC.scm` functions, which redefine functions defined in `ARSdefault.scm`, are identified as "overriding the standard."

`add-alphalist-object` *object* — *C* Function

Adds the specified *object* to the list of objects displayed in the alpha list. Subsequent calls to display the alpha list will display the object in the alpha list even though the object may not be on the selectlist.For example:

```
(add-alphalist-object
  (find-sole-arrival))
```

`add-and-save-frozen-object-remarks <arrival` *object* or `origin` *object*`>` — *C* Function

Provides a means of adding remarks to an arrival or origin that has already been frozen (saved to the database).

`add-assoc-error` — `IDC.scm`

Adds strings to the *assoc* error message and stores the erroneous objects in lists so that they will be selected upon completion of the checking process.

Calling objects:
`check-assoc-residual`
`add-residual-error`

Does not override the standard *Scheme* function.

`add-best-channels` ARSdefault.scm

Adds to the display the additional channels that are listed in the variable `*best-channels*`, which is also defined in `ARSdefault.scm`.

`add-button` *location-str label Scheme-exp* *C* Function

Adds a graphic button in the specified location labeled with *label*. When the button is pressed, *Scheme-exp*, which is a string, is submitted to the *Scheme* interpreter. *location-str* must be either "`Toolbar`", "`Alphalist`" or "`Locator`".

`add-menu-item` *menu item Scheme-exp* *C* Function

Adds a menu item to *menu* with name *item.* When the *item* is selected, *Scheme-exp* is submitted to the interpreter. All arguments are strings.

`add-multiple-filters` ARSdefault.scm

Prompts the user for several lines of text. The response is added to the list of filters.

`add-multiple-remarks` ARSdefault.scm

Prompts the user for multiple lines of arbitrary remarks. The response is added to the remarks for each object in the current selection list.

`add-origin-error` *origin text* IDC.scm

Adds strings to the origin error message. It stores the erroneous objects in lists so that they will be selected upon completion of the checking process.

Calling object: `check-origin-for-two-p-phases`

Does not override the standard *Scheme* function.

`add-residual-error` *assoc name colname* `IDC.scm`

Used in conjunction with `(add-assoc-error)` when an attribute is incorrectly set as a defining attribute, this function appends an error message to the *assoc* and sets the erroneous attribute to nondefining.

Calling object: `check-assoc-residual`

Does not override the standard *Scheme* function.

`add-selectlist-arrivals-all` `ARSdefault.scm`

Adds all arrivals to the selection list.

`add-selectlist-arrivals-in-window` `ARSdefault.scm`

Adds all arrivals in the displayed time window to the selection list. When the waveform window is aligned, the time window is between the earliest time on the earliest displayed channel and the latest time on the latest displayed channel.

`add-selectlist-associated-arrivals-with-phases` *phase-list*

`ARSdefault.scm`

Adds to the selection list those arrivals associated to the selected origin whose phases exist in the given list of phases, *phase-list*.

`add-selectlist-associated-arrivals-without-phases` *phase-list*

`ARSdefault.scm`

Adds to the selection list those arrivals associated to the selected origin whose phases do not exist in the given list of phases, *phase-list*.

`add-selectlist-associated-coda-arrivals` `IDC.scm`

Adds to the selection list those arrivals that are associated to the selected origin and have coda phases (specified in `list-of-coda-phases`).

Calling object: `select-coda-and-disassociate`

Does not override the standard *Scheme* function.

`add-selectlist-associated-origins` `ARSdefault.scm`

Adds all origins that are associated with the selected arrivals to the selection list.

`add-selectlist-associated-other-arrivals` `IDC.scm`

Adds to the selection list those arrivals that are associated to the selected origin and that are not coda phases (specified in `list-of-coda-phases`).

Does not override the standard *Scheme* function.

`add-selectlist-associated-stassocs` `ARSdefault.scm`

Adds all *stassocs* that are associated with the selected arrivals to the selection list.

`add-selectlist-channels-all` `ARSdefault.scm`

Adds all channels to the selection list.

`add-selectlist-component-channels` *component-name* `ARSdefault.scm`

Clears the selection list, then adds only the channels of a given component. For example:

```
(add-selectlist-component-channels "cbP")
```

`add-selectlist-derived-channels` `ARSdefault.scm`

Clears the selection list, then adds only the channels that are derived.

`add-selectlist-object` *object* *C* Function

Adds *object* to the selection list. The following example adds the first arrival on the arrival list to the selection list:

```
(add-selectlist-obj
  (car (say arrivals)))
```

`add-selectlist-origin-associated-arrival` *origin* `ARSdefault.scm`

Adds all arrivals that are associated with a specified origin to the selection list.

`add-selectlist-origins-all` `ARSdefault.scm`

Adds all origins to the selection list.

`add-selectlist-origins-associated-arrivals` `ARSdefault.scm`

Adds all arrivals that are associated with any origins on the selection list.

`add-selectlist-stassoc-associated-arrivals` *stassoc* `ARSdefault.scm`

Adds all arrivals associated with a specified *stassoc* to the selection list**.**

`add-selectlist-stassocs-all` `ARSdefault.scm`

Adds all *stassocs* to the selection list.

`add-selectlist-stassocs-associated-arrivals` `ARSdefault.scm`

Adds all arrivals associated with any origins on the selection list.

`add-selectlist-user-associated-arrivals` `IDC.scm`

Adds to the selection list those arrivals in the user's phase list that are associated to the selected origin.

Does not override the standard *Scheme* function.

`add-single-filter` `ARSdefault.scm`

Prompts the user for filter parameters. The input is added to the list of filters.

`add-to-help-list` *function-name help-string* C Function

Adds a documentation *help-string* for a specified function to the online help database. The documentation string is displayed when the "Menu Help" is selected, and a function is designated. The *function-name* is a string and should be enclosed in parentheses. For example:

```
(add-to-help-list
  "(my-new-function)"
  "This is only a test function.")
```

`add-to-period` *period offset* `ARSdefault.scm`

Increments or decrements a time period. *period* is a two-element list of ordered times. A new period is returned with the early time decremented by *offset* and the later time incremented by *offset*.

`align-channel-on-phase` *channel phase* `IDC.scm`

Aligns a single *channel* to a *phase* given an origin.

`align-channels` *channel-list origin phase try-designated* C Function

Changes the display start time for all channels in *channel-list* so that the theoretical arrival time of *phase* from *origin* at the station is aligned with the first channel's theoretical *phase* arrival. If *try-designated* is true, the alignment is on the designated arrival if possible, else it is on the theoretical arrival.

`align-channels-on-designated-phase` *phase* `ARSdefault.scm`

Calls `align-channels-on-phase` with `True` so that `align-channels` will attempt to align on the designated *phase* first, then the theoretical phase.

`align-channels-on-initial-phase` C Function

Aligns channels on the measured P, PKPdf, Pn, Pg, or Pdiff phase. The phases are sought in the order given; if none of the phases exist, the theoretical times are used.

`align-channels-on-theoretical-phase` *phase* `ARSdefault.scm`

Calls `align-channels-on-phase` with nil (`False`) so that `align-channels` will align strictly on theoretical phases.

`align-channels-on-phase` *phase try-designated period* `ARSdefault.scm`

Aligns all channels to the theoretical arrival time of *phase* from the origin on the selection list. Only one origin may be selected. The travel-time tables are initialized if necessary.

`align-displayed-on-i` `IDC.scm`

Creates a list of all displayed channels and passes the list as an argument to `align-many-on-i`.

Does not override the standard *Scheme* function.

`align-displayed-on-t` `IDC.scm`

Creates a list of all displayed channels and passes the list as an argument to `align-many-on-t`.

Does not override the standard *Scheme* function.

`align-hydro-on-t` `IDC.scm`

Creates a list of all hydro channels and passes the list as an argument to `align-many-on-t`.

Does not override the standard *Scheme* function.

`align-infra-on-i` `IDC.scm`

Creates a list of all infra channels and aligns on the “I” phase.

Does not override the standard

`align-many-on-i` *chanlist* `IDC.scm`

Recursive function that aligns each channel in its list on phase “I”.

Does not override the standard *Scheme* function.

`align-many-on-t` *chanlist* `IDC.scm`

Recursive function that aligns each channel in its list on phase "T".

Does not override the standard *Scheme* function.

`align-selected-on-i` `IDC.scm`

Creates a list of all selected channels and passes the list as an argument to `align-many-on-i`.

Does not override the standard *Scheme* function.

`align-selected-on-t` `IDC.scm`

Creates a list of all selected channels and passes the list as an argument to `align-many-on-t`.

Does not override the standard *Scheme* function.

`alpha-assoc` `IDC.scm`

Displays an alpha list with all associated arrivals, then deselects arrivals.

Does not override the standard *Scheme* function.

`alphalist-is-shown?` *C* Function

Predicate function that returns `t` or `nil` depending upon whether or not the alpha list is displayed.

`append-cascade-filter` *flist* `ARSdefault.scm`

Given a list of filter strings, *flist*, this function creates a cascade filter string and appends it to `list-of-cascade-filters`. The list is sorted, and duplicates are attempted to be removed, but the function does not recognize linear equivalents (for example, A->B == B->A).

`append-true-stations` *arrivals* `ARSdefault.scm`

Takes a list of *arrivals* and returns a list of (arrival true-station) elements. A true station is the true station name for a group of elements; for example, the true station for NRA0 is NORES. The true station name should be the name of the network to which an element belongs in the affiliation table. Although this table is often **refsta**, it may not be, which is why this function exists. All station attributes (such as *reliability*) are keyed to the true station, so this mapping must occur before processing. For example, `(`*arr1 arr2 ...*`)` returns `((`*arr1 sta1*`) (`*arr2 sta2*`)` *...*`)`.

`apply-scanning-filters` *chans* `ARSdefault.scm`

Applies special scanning filters to the channels by using the function `(find-scanning-filter-for-channel)`.

`arrival?` *object* *C* Function

Returns `t` if *object* is an arrival and `nil` if it is not.

`arrival-in-time-interval?` *station channel time duration* *C* Function

Determines if arrivals are within the *time* period on the specified *station* and *channel*. The following example returns `t` if an arrival exists within the time period "3/06/90 13:00:00" to "3/06/90 13:01:00" on the station and channel corresponding to the first displayed channel:

```
(arrival-in-time-interval?
  (extract-channel-station
  (car (say-channels t)))
  (extract-channel-channel
  (car (say-channels t)))
  (human-time>epoch-time ("3/06/90 13:00:00") 60.0))
```

`ars-auto-write` *C* Function

Writes *ARS* internal data structures to a flat file for use in application crash recovery.

`*assoc-check-functions*` `IDC.scm`

Runs the following individual functions:

`(check-assoc-time-residual)`

`(check-assoc-azimuth-residual),`

`(check-assoc-slowness-residual)`

Calling object: `check-origin-list`

Does not override the standard *Scheme* function.

`associate-origin-arrival` *arrival origin* *C* Function

Associates the given *arrival* with the given *origin*.

`associated-origin-arrival?` *arrival* *C* Function

Predicate function that returns `t` if *arrival* is associated with an origin.

`associate-origin-selected-arrivals` `ARSdefault.scm`

Associates all arrivals on the selection list with one origin on the selection list.

`associate-stassoc-arrival` *arrival stassoc* *C* Function

Associates the given *arrival* with the given *stassoc*.

`associated-stassoc-arrival?` *arrival* *C* Function

Predicate function that returns `t` if *arrival* is associated with an origin.

`associate-stassoc-selected-arrivals` `ARSdefault.scm`

Associates all arrivals on the selection list with one *stassoc* on the selection list.

`beam-p-phase` `IDC.scm`

Sends a beamer message by using the default P phase and returns *ARS* to an active state without waiting for the returned beam.

Does not override the standard *Scheme* function.

`begin-sync` *list env* — *C* Function

Executes the *LISP* list of *Scheme* functions, synchronizing (with *SyncXt C* call) the graphics between each function call.

`broken?` *object* — *C* Function

Returns `t` if *object* is broken. Currently, only origins may be broken: An origin has one or more associated arrivals removed, breaking its location hypothesis.

`calculate-median` *num-lst* — `ARSdefault.scm`

Calculates the median value of a list of numbers, *num-lst*. If the number of elements is even, this function takes averages the two "center" elements. It returns `nil` if the list is nil.

`channel?` *object* — *C* Function

Returns `t` if the *object* is a channel object or `nil` if it is not. The following example tests if the first object on the selection list is a channel:

```
(channel? (car (say-selectlist))
```

`channel-in-origin-time-interval?` *chan orig* — `ARSdefault.scm`

Checks if a particular channel, *chan*, has data around the expected P arrival time for an origin, *orig*.

`channel-in-time-interval?` *channel begin-time duration* — `ARSdefault.scm`

Checks if a *channel* has a waveform within a given time interval. This routine will always return `nil` for a channel that has never been displayed.

`channel-same-chan` *chan chanlist* — `ARSdefault.scm`

Returns a list of undisplayed channels with the same channel.

`channel-same-sta` *chan chanlist* — `ARSdefault.scm`

Returns a list of undisplayed channels with the same station.

The following 38 `check-*` functions support the `(confirm-event)` function; they assess events to determine conformity with processing guidelines. These functions create and return lists of strings that describe potential variances. In these functions, the input argument is a list of associated detections such as those returned from

```
(find-origin-associated-arrivals (find-sole-origin))
```

Errors result in an arid/error pair. The list of pairs is returned, for example:

`((`*arid1* `"`*Error string 1*`")` `(`*arid2* `"`*Error string 2*`")` ...`)`

`check-and-create-arrival` *phase-to-add* `IDC.scm`

Replaces `(prompt-phase-create-arrival)` to support the new check for existing arrivals within four seconds.

Calling object: `prompt-phase-create-arrival`

Does not override the standard *Scheme* function.

`check-and-retime-arrival` `IDC.scm`

Checks that the user has not retimed the arrival more than four seconds. It does not check for multiple retimes, that is, if the user retimes an arrival for three seconds, then this function retimes the same arrival another three seconds; the net six-second retime is allowed.

Does not override the standard *Scheme* function.

`check-arrival-multiple-association` *stassocs* `ARSdefault.scm`

Checks that each arrival has only one origin association. The error list is in the following form:

`((`*arid1* `"Station <`*sta*`>, arid <`*arid*`> has multiple associations in orids <`*orid1*`>, <`*orid2*`>, <`*oridn*`>")` ...`)`

`check-assoc-azimuth-residual` *assoc* IDC.scm

Uses `(check-assoc-residual)` and `*phase-azres-table*` to determine if the residual in the passed *assoc* object is acceptable. If it exceeds the limits set in the table, an error message is appended to the *assoc* error text.

Calling object: `*assoc-check-functions*`

Does not override the standard *Scheme* function.

`check-assoc-residual` *assoc table fieldname* IDC.scm

Checks the residual value of the *fieldname* in *assoc* against the limit specified in the *table*.

Calling objects:
`check-assoc-slowness-residual`
`check-assoc-time-residual`
`check-assoc-azimuth-residual`

Does not override the standard *Scheme* function.

`check-assoc-slowness-residual` *assoc* IDC.scm

Uses `(check-assoc-residual)` and `*phase-timeres-table*` to determine if the residual in the passed *assoc* object is acceptable. If it exceeds the limits set in the table, an error message is appended to the *assoc* error text.

Calling object: `*assoc-check-functions*`

Does not override the standard *Scheme* function.

`check-assoc-time-residual` *assoc* IDC.scm

Uses `(check-assoc-residual)` and `*phase-slores-table*` to determine if the residual in the passed *assoc* object is acceptable. If it exceeds the limits set in the table, an error message is appended to the *assoc* error text.

Calling object: `*assoc-check-functions*`

Does not override the standard *Scheme* function.

`check-associated-hydro-blockage` *plot-map* `ARSdefault.scm`

Main function for blockage checking.

`check-defining` *assoc type* `IDC.scm`

Determines whether or not the phase is defining for the given *assoc*. If it is, the function verifies that it has not been forced as nondefining because of a previous error.

Calling objects:
`check-origin-for-two-p-phases`
`check-assoc-residua`

Does not override the standard *Scheme* function.

`check-defining-measure-is-allowable` *stassocs* `ARSdefault.scm`

Checks that each defining measurement is allowed to be defining for the given phase, as given in `*defining-phase-residuals-list*`. Returns a list of all errors in the following form:

`((`*arid1* `"Station <`*sta*`>, arid <`*arid*`>, phase <`*phase*`> should not have a defining <`*type*`> measurement.") ...)`

`check-defining-phase-rules` *associations* `ARSdefault.scm`

Checks defining phase rules and guidelines [Bow95]: (1) A primary P-type phase must be the first associated defining at each station. (2) A teleseismic secondary phase may only be defining if the preceding primary phase is defining. (3) A phase may be defining for only one event. (4) A teleseismic defining phase must have a defining arrival time. (5) A phase measurement type may only be defining if it has a non-null residual limit.

`check-defining-unreliable-slowness-vector` *associations*
`ARSdefault.scm`

Checks the unreliable slowness vector guideline [Bow95]: Azimuth and slowness should be nondefining for stations for which slowness vector estimates are not reliable.

`check-event-depth-for-hydro-event` *org associations* `ARSdefault.scm`

Checks that if an event is comprised of only hydroacoustic arrivals, the depth is less than a given threshold.

`check-first-phase-primary` *stassocs* `ARSdefault.scm`

Checks a time-sorted list of defining station associations and verifies that the first phase is a primary P-type phase. If not, the function returns a list of the following form:

`((-1 "Station <`*sta*`> has the nonprimary phase <`*phase*`> as its first defining phase."))`

`check-for-P-GT-103` *associations* `ARSdefault.scm`

Checks for `"P"` phases at Pdiff/PKP distances.

`check-for-allowable-depth-uncertainty` *orig* `ARSdefault.scm`

Checks if an origin has allowable depth uncertainty. If it does not, the function returns a list of the following form:

`((-1 "Origin <`*orid*`> has an unallowable depth uncertainty of <`*val*`>") ...)`

The `-1` value specifies to further routines that this error is origin based, not arrival based. Maximum depth uncertainty is defined as follows (see [Bow95]):

If *depth* < 5 km, *max-depth-uncertainty* = 10 km.

If *depth* 5 km < *depth* <= 100 km, *max-depth-uncertainty* = 2 * *depth*.

If *depth* > 100 km, *max-depth-uncertainty* = 200 + (0.5 * (*depth* - 100)).

`check-for-allowable-magnitude` *orig* `ARSdefault.scm`

Checks if an origin has allowable magnitude (null is acceptable). If it does not, the function returns a list of the following form:

`((-1 "Origin <`*orid*`> has an unreasonable magnitude value of of <`*val*`>") ...)`

The `-1` value specifies to further routines that this error is origin based, not arrival based.

`check-for-allowable-phase-distance-ranges` *associations*

`ARSdefault.scm`

Checks if the associated phases are within their proper ranges. Phase ranges are obtained from the global structure `*defining-phase-residuals-list*`.

`check-for-allowable-phases` *associations* `ARSdefault.scm`

Checks all associated arrivals to verify that the phase is in the set of allowable phases. If it is not, the function adds the arrival to the error list to be returned. The error list is in the following form:

`((arid1 "Station` *<sta>* `arid` *<arid1>* `has an improper defining phase` *<phase>*`") ...)`

`check-for-allowable-residuals` *associations* `ARSdefault.scm`

Checks all associated arrivals to verify that each measurement residual is reasonable. If it is not, the function adds the arrival to the returned error list, which has the following form:

`((arid1 "Station` *<sta>* `arid` *<arid1>* `has a` *<type>* `residual of` *<val>*`") ...)`

`check-for-amplitude-consistency` *orig associations* `ARSdefault.scm`

Checks the amplitude consistency rule [Bow95]: "The amplitude of a defining, first-arriving phase should be consistent with the same phase at other stations at similar distance range, unless one station is near a caustic. Allowing for variations in radiation pattern and efficiency of propagation, amplitudes should generally be consistent to within two orders of magnitude." This routine checks the value of station magnitudes against the median value of the station magnitudes. If the station magnitude differs by more than 1.0 (originally 0.7, but changed in June 1995) magnitude units from the median and the station, an error message is appended to the error list.

`check-for-az-sl-define` *associations* `ARSdefault.scm`

Checks for errors in the case when a non-time-defining association is set to be azimuth and/or slowness defining.

`check-for-event-depth-and-regional-phases` *orig associations*
`ARSdefault.scm`

Checks regional-type phases associated with an event to verify that the event depth is reasonable (depth - sdepth < 40 km) [Bow95].

`check-for-large-event-defining-rules` *associations* `ARSdefault.scm`

Checks if the associated arrivals for a large event (one with more than six stations with primary phases) violates the guidelines in the large event rule [Bow95]: For large events with P-type phases at six or more stations, azimuth and slowness should be nondefining and later phases (other than depth-sensitive phases) should be associated but nondefining. Secondary regional phases are acceptable.

`check-for-minimum-alpha-stations` *associations* `ARSdefault.scm`

Checks if an event has at less than `*minimum-alpha-stations*` stations. This list is in the following form [Bow95]:

((-1 "Event has *<num>* Alpha stations. Below minimum of *<num>*."))

`check-for-origin-uncertainty` *origin* `ARSdefault.scm`

Checks if the major-axis of the origin is greater than threshold.

`check-for-regional-phase-names` *associations* `ARSdefault.scm`

Checks if regional phases are declared for stations at teleseismic distances.

`check-for-regional-slowness` *associations origin* `ARSdefault.scm`

Checks if regional phases are declared for stations at teleseismic distances.

`check-for-small-event-defining-rules` *associations* `ARSdefault.scm`

Checks alpha stations for conformance with the small event rule [Bow95]: For small events with P-type phases at <= 5 stations, azimuth and slowness should be defining, if measurements are within the residual limits.

`check-for-tele-phase-names associations` *origin* `ARSdefault.scm`

Checks if teleseismic phases are declared for stations at regional distances.

`check-for-unallowed-defining-phases` *associations origin* `ARSdefault.scm`

Checks for time-defining phases that are not members of the defining phases list.

`check-if-good-for-hydro-display` `IDC.scm`

Displays a message that indicates if hydro analysis is required by the current rules.

Does not override the standard *Scheme* function.

`check-origin-for-two-p-phases` *origin* `IDC.scm`

Checks if the event has time-defining P phases on at least two stations.

Calling object: `*origin-check-functions*`

Does not override the standard *Scheme* function.

`check-origin-list` *origin-list popup* `IDC.scm`

Checks the origin to ensure it has two time-defining P phases on at least two stations. The function checks the associated arrivals to see if the time residuals, azimuth residuals, and slowness residuals are within specified bounds. Erroneous objects are added to the selection list so they can be viewed by the alpha list. If `popup-p` is `nil`, no output will be printed; if it is `t`, output will be printed in a popup window. When called from `(read-travel-time-tables-and-locate)` no output is printed.

Calling object: `read-travel-time-tables-and-locate`

Does not override the standard *Scheme* function.

`check-secondaries-have-preceding-primary` *stassocs* `ARSdefault.scm`

Checks a time-sorted list of defining station associations and verifies that associations with a secondary phase type follow a primary phase association. If not, the function returns a list in the following form:

`((arid1 "Station` *<sta>*`, arid` *<arid>* `with secondary phase` *<phase>* `does not follow a defining primary phase.") (arid2 "Station` *<sta>*, `arid` *<arid>* `with secondary phase` *<phase>* `does not follow a defining primary phase.")` ... `)`

`check-selected-stations-for-hydro-blockage` *plot-map*

`ARSdefault.scm`

This function is similar to `check-associated-hydro-blockage` except that selected hydroacoustic stations are checked for clear paths to the selected origin, and the stations do not have to have associated hydroacoustic arrivals.

`check-slowness-vector-consistency` *associations* `ARSdefault.scm`

Checks for conformance with the slowness-vector-consistency guideline [Bow95]: An arrival time from a station with a high-quality slowness vector observation can only be defining if the slowness and azimuth residuals are less than two times the limits shown in `*defining-phase-residuals-list*`. This guideline applies only to stations for which the automatic processing system produces reliable slowness-vector estimates (see `*station-azimuth-slowness-reliability-list*`).

`check-time-defining` *stassocs* `ARSdefault.scm`

Checks a list of defining station associations and verifies that teleseismic associations are time defining. If errors occur, the function returns a list in the following form:

`((arid1 "Station` *<sta>*`, arid` *<arid>* `is teleseismic, but is not time-defining."))`

`close-alpha-list` *C* Function

Closes the alpha list window.

`compare-sta-chan-distance-priority` *obj1 obj2* `IDC.scm`

Determines the relative sorting priority of two stations based on their event distance, station alphabetic order, and the channel priority. A single origin must be selected.

Calling object: `new-sort-selected-channels`

Does not override the standard *Scheme* function.

`compress-window` `IDC.scm`

Expands the time duration of data that is displayed, currently by a factor of two.

Does not override the standard *Scheme* function.

`compress-window-by8` `IDC.scm`

Expands the time duration of data that is displayed, currently by a factor of eight.

Does not override the standard *Scheme* function.

`compute-angular-distance` *location1 location2* — *C* Function

Computes the great circle distance in degrees between two points (*location1*, *location2*) on the earth. Locations are three-element lists '(*lat lon depth*)'. Depth is ignored in the distance calculation. The following example computes the angular distance from the point (10.0 10.0 5.0) to (20.0 20.0 5.0).

```
(compute-angular-distance '(10.0 10.0 5.0) '(20.0 20.0 5.0))
```

`compute-delta-seaz-esaz` — *C* Function

Computes the delta, azimuth, and back azimuth between two locations, each of which is specified as a list of latitude, longitude, and depth. A list of the delta, azimuth, and back azimuth is returned.

`compute-sta-event-delta` *chan origin* — `ARSdefault.scm`

Computes the angular distance between a given station and origin.

`compute-sta-prob` *delta depth snr-thresh noise sd-noise mag atten-file* — *C* Function

Computes the detection probability for a given event at a given station. The arguments are defined as follows:

delta	event/station distance in degrees
depth	event depth
snr-thresh	detection snr threshold at station
noise	characteristic noise level at station
sd-noise	characteristic noise (log standard deviation level at station)
mag	event magnitude
atten-file	path to attenuation file

`compute-selectlist-arrivals-vector-slores` *list* `ARSdefault.scm`

Returns a list of lists for all arrivals on the selection *list*. The returned list consists of the *arids* (represented as a float) and the magnitudes of the slowness vector residual for each arrival on the selection list, for example:

```
((36824.0000 6.1384) (36822.0000 () ) (36823.0000 1.2346))
```

The slowness vector residual is computed by using the Law of Cosines, defined as follows:

$$\sqrt{(\mathrm{slow}^2 + (\mathrm{slow} - \mathrm{slores})^2) - (2|\mathrm{slow} - \mathrm{slores}|\cos(\mathrm{azres}))}$$

`compute-travel-time` *location1 location2 phase* *C* Function

Computes the expected travel time in seconds between *location1* and *location2* for a *phase*. Travel-time tables must be read before this function is called. A location is a three-element list containing `'(`*lat lon depth*`)'`. The following example computes the expected travel time between the single, selected channel and the single, selected origin for phase `“LR”`.

```
(compute-travel-time
  (extract-channel-location
  (find-sole-channel))
  (extract-origin-location
  (find-sole origin)) “LR”)
```

`concat` *lst* `IDC.scm`

Converts a list of strings into a single string with carriage returns.

Calling object: `show-remarks-box`

Overrides standard *Scheme* function, but in `ARSdefault.scm` this function is local to `show-remarks-box`.

`concat-filter-strings` *chosen-list* `ARSdefault.scm`

Concatenates a given list of strings into a single list with the strings separated by `"/"`. For example, (`"a" "b" "c"`), is concatenated to `"a/b/c"`. This function is used for creating a string for cascaded filters. The `"/"` is a key character that is recognized in *DoFilter()*.

`confirm-event` ARSdefault.scm

Checks a number of event attributes to verify event correctness. These attributes include picked phases, residual values, magnitude values, and a weighted count for an event's associated arrivals. This count is a function of the defining features of the associated arrivals.

`copy-and-write-beam` *channel object chan-string* *C* Function

Copies the given channel containing beam data as either an origin or arrival beam, writes the data to the directory specified by `perm-wfdisc-directory`, updates the **wftag** and **wfdisc** tables, and displays the new beam channel. The arguments are defined as follows:

channel	channel object from which the data are copied
object	origin or detection object
chan-string	name string to use for the new beam channel

`copy-channel` *channel* *C* Function

Makes a working copy of a single *channel*.

`copy-channels` ARSdefault.scm

Makes duplicate working copies of the channels on the selection list.

`count-if` *cond-func lst* ARSdefault.scm

Counts all elements from a list that satisfy a condition function.

`create-aeq-message` *origin* ARSdefault.scm

Creates an interprocess communication (IPC) message for the Anomalous Event Qualifier (AEQ).

`create-alpha-list` *channels origins stassocs arrivals* — *C* Function

Creates and displays an alphanumeric list of the specified *channels*, *origins*, *stassocs*, and *arrivals*. The following example displays an alpha list with all selected origins. The *channel*, *stassoc*, and *arrival* lists are empty lists.

```
(create-alpha-list ()
  (say-selected-origins) () ())
```

The "Alpha List" menu item invokes the following command:

```
(create-alpha-list
  (say-selected-channels)
  (say-selected-origins)
  (say-selected-stassocs)
  (say-selected-arrivals))
```

`create-and-send-XfkDisplay-message` *arrival form-beam*
`ARSdefault.scm`

Creates and sends a message to *XfkDisplay* for an arrival object. If *form-beam* is non-null, a detection beam will be created by *XfkDisplay* and will be displayed automatically in *ARS*.

`create-and-send-map-arc-message` *lat1 lon1 lat2 lon2 backpath color*
`ARSdefault.scm`

Sends the map application the coordinates to plot an arc with proper color and backpath string.

`create-and-send-map-channel-message` *channel* — `ARSdefault.scm`

Sends the map application a message to plot a channel.

`create-and-send-map-origin-message` *origin* — `ARSdefault.scm`

Sends the map application a message to plot an origin.

`create-arrival` *phase channel time* — *C* Function

Creates a new arrival at *time* on *channel* with the name *phase.* The following example creates a new arrival on the single, selected channel with phase `“Pn”` and time `“03/06/90 13:14”`.

```
(create-arrival “Pn”
  (find-sole-channel)
  (human-time->epoch-time
  “03/06/90 13:00:14))“
```

`create-arrival-with-phase` *phase* — `IDC.scm`

Creates a new arrival identified as the seismic *phase* selected by a user. This function is intended to be invoked by `prompt-phase-name`. If a single origin is on the selection list, the new arrival will be associated with the origin.

Calling object:
`check-and-create-arrival`
`prompt-phase-create-arrival`

Overrides the standard *Scheme* function. The override supports the proper setting of time, slowness, and azimuth defining attributes.

`create-beamer-message` — `ARSdefault.scm`

Creates the message string that will be sent to the *Beamer* program to build waveform data. This function creates some internal data structures that are de-allocated upon receiving the reply message.

`create-channel-menu` *chans* — `ARSdefault.scm`

Creates a channel menu from a list of channels.

`create-dfx-hydro-ipc-info` — `ARSdefault.scm`

Calls the function to populate the hydro_features temporary table and then builds the IPC string needed to send from *ARS* to *DFX* for interactive hydroacoustic recall processing. This function has no arguments and returns an ipc-string.

`create-dfx-recall-message` ARSdefault.scm

Creates the message string that will be sent to the *DFX* program for recall processing. This function gets called only from `(send-receive-dfx-recall-message)`.

`create-gaim-message` *origin* ARSdefault.scm

Creates the message string that will be sent to the *GAim* program. This function gets called only from `(send-receive-gaim-message)`.

`create-filter-menu` *filtlist* ARSdefault.scm

Creates a filter menu from a list of filters.

`create-idc-dfx-recall-message` ARSdefault.scm

Creates the message string that will be sent to the *DFX* program for recall processing for the IDC. This function gets called only from `(send-receive-dfx-recall-message)`.

`create-idcarrival-with-phase` *phase* ARSdefault.scm

Prompts the user for a phase name and creates a new arrival with the returned phase name at the time t1. This function is a derivative of `(prompt-phase-create-arrival)`, but has *timedef*, *slodef*, and *azdef* set to IDC values. Px, Sx, and tx phases are not associated, even if an origin is selected.

`create-interclass-message` *C* Function

Creates the message string that will be sent to the *InterClass* program, which classifies events. This funtion gets called only from `(send-inter-class-message)`.

`create-libpar-selectlist` ARSdefault.scm

Creates a list containing objects on the selectlist in libpar format.

`create-libpar-ipc-message` *list* — *C* Function

Takes a *list* of lists and returns a string suitable for libpar parsing. Internal lists have the format *(name type (value1 value2 value3...))* with *type* determining the need for single quotes surrounding the *value(s)*; an infinite number of internal lists may result. The returned string has the following format:

name = value1, value2, value3

`create-libpar-strings` *message-list* — *C* Function

Takes an internal *message-list* in libpar format, determines if it has any valid members, and if so, determines the message type.

`create-libpar-anytype-string` *list* — *C* Function

Takes a libpar format *list* and returns a libpar format string by appending pieces of the *list* with an equal sign (=) and spaces.

`create-libpar-strtype-string` *list* — *C* Function

Takes a libpar format *list* and returns a libpar format string by appending pieces of the *list* with an = sign, single quotes, and spaces.

`create-long-period-arrival` — *C* Function

Unselects all arrivals, creates and selects an arrival, performs automatic amplitude/period measurement, and retimes the arrival so it is midway between the two amp/per measurement points.

`create-message-string` *number-of-lists list* — *C* Function

Creates a message string suitable for IPC message passing, given a *list* of lists with elements to be included in the string.

`create-origin` — *C* Function

Creates a new origin object. New origins automatically are assigned a temporary *orid* that is less than 0; they are assigned permanent *orids* when they are saved to the database.

`create-stassoc` C Function

Creates a new *stassoc* object. New *stassocs* are automatically assigned a temporary *stassid* that is less than 0; they are assigned permanent *stassids* when saved to the database.

`create-theoretical` *phase channel time* C Function

Creates a theoretical arrival with the specified *phase* name on a given *channel* at a given *time*.

`current?` *object* C Function

Tests if the location hypothesis of an *object* (for example, origin) is consistent with the currently associated arrivals. The *object* is considered modified (and not current) if an associated arrival has been added, deleted, retimed, or renamed since the location hypothesis was computed. The following example returns `t` if the selected origin is current:

```
(current? (find-ole-origin))
```

`defining-association?` *association* `ARSdefault.scm`

Returns a non-nil value if arrival is defining.

`delete-arrival` *arrival* C Function

Deletes an unfrozen *arrival*. The following example deletes the single, selected arrival.

```
(delete-arrival (find-sole-arrival))
```

`delete-arrivals` `ARSdefault.scm`

Maps `delete-arrivals` onto the list of arrivals on the selection list. `delete-arrival` deletes an unfrozen arrival.

`delete-button` *location-str label* C Function

Deletes the graphic button specified by *location-str* and *label* (see `add-button`).

`delete-confirm-arrivals` `ARSdefault.scm`

Deletes the arrivals on the selection list after confirmation from the user. When executed, the arrivals are displayed in a preselected list. The user deselects the arrivals at will, then selects "Done." The arrivals that remain selected are deleted. "Cancel" does not delete any arrivals.

`delete-confirm-origins` `ARSdefault.scm`

Deletes the origins on the selection list after confirmation from the user. When executed, the origins are displayed in a preselected list. The user deselects the origins at will, then selects "Done." The origins that remain selected are deleted. "Cancel" does not delete any origins.

`delete-disassociate-origin` `ARSdefault.scm`

Disassociates all arrivals from the sole selected origin and deletes the origin from the origin display area. The selected origin must be the only one on the selection list and cannot be frozen.

`delete-menu-item` *menu item* C Function

Deletes the menu item in *menu* with name *item* (see `add-menu-item`).

`delete-origin` *origin* C Function

Deletes the specified *origin* from the internal database if the *origin* is unassociated with any arrivals. For example:

```
(delete-origin (find-sole-origin))
```

`delete-remarks` *obj* C Function

Deletes the remarks associated with the specified *object*.

`depth-phase?` *phase* `ARSdefault.scm`

Returns a non-nil value if phase is a depth-phase such as pP or pPKPbc.

`depth-sensitive-phase?` *phase* `ARSdefault.scm`

Returns a non-nil value if phase is a depth-sensitive-phase such as PcP or ScP.

`derived?` *object* — C Function

Predicate function that determines whether an *object* is derived from other data. Currently, only theoretical arrivals and copied channels are derived. This function returns `t` if the *object* is derived or `nil` if it is not. The following example returns `t` if the single selected arrival was derived:

```
(derived? (find-sole-arrival))
```

`disassociate-origin-arrival` *arrival* — C Function

Disassociates *arrival* from its associated origin. The following example shows no indication that the arrival actually had an associated origin.

```
(disassociate-origin-arrival
  (find-sole-arrival))
```

`disassociate-origin-selected-arrivals` — `ARSdefault.scm`

Disassociates all arrivals on the selection list from their associated origins.

`disassociate-stassoc-arrival` *arrival* — C Function

Disassociates *arrival* from its associated *stassoc*.

`disassociate-stassoc-selected-arrivals` — `ARSdefault.scm`

Disassociates all arrivals on the selection list from their *stassocs*.

`discard-origin` — `IDC.scm`

Prompts for a list of reasons, then adds the sole selected origin and the selected reasons to the **discard** table.

`discarded?` — C Function

Predicate function that returns `t` if the sole selected origin exists in the **discard** table.

`display-and-filter-hydro-arrivals` *object display-flag* — *C* Function

Displays and filters or undisplays and unfilters the onset/termination time widgets for hydro arrivals. Returns `t` or `nil` indicating the requested action. *object* is the detection object; *display-flag* is the filter; display is (`t`), or unfilter and undisplay is `(nil)`.

`display-and-map-associated-hydro-blockage` — `ARSdefault.scm`

Checks all associated stations having hydroacoustic phases for blocked water paths, prints a list of blocked stations, and plots the event-to-station paths on the map.

`display-and-map-selected-stations-hydro-blockage` — `ARSdefault.scm`

Checks all selected hydroacoustic stations for blocked water paths, prints a list of blocked stations, and plots the event-to-station paths on the map.

`display-associated-hydro-blockage` — `ARSdefault.scm`

Checks all associated stations having hydroacoustic phases for blocked water paths and prints a list of blocked stations.

`display-blocked-channels-message` *selected-origin blocked-chans*
`ARSdefault.scm`

Displays a message listing the predicted blocked hydro stations for a particular origin.

`display-blocked-phases-message` *origin blocked-dets* — `ARSdefault.scm`

Displays a message listing the predicted blocked hydro phases for a particular origin.

`display-confirm-info` *wt-cnt error-list orig* — `ARSdefault.scm`

Prints confirmation errors and weighted count. *error-list* is in the form `((`*arid* `"error string)` `(`*arid* `"error string)` ...`)` where *arid* = `-1` if the error is origin based.

`display-selected-stations-hydro-blockage` `ARSdefault.scm`

Checks all selected hydroacoustic stations for blocked water paths and prints a list of blocked stations.

`edit-eval-string` *prompt function string* *C* Function

Pops up an edit text box from which the user may view and edit a single *string*. The text box is labeled with *prompt* and is initialized with *string*. When the user is finished editing and presses the "Done" button, *function* is called with the edited *string* as an argument.

The following example prompts the user with a single-line text editing box labeled `"Test Me"`. The box is initialized with the string `"Hello how are you?"`. If the user edits the string and presses the "Done" button, the value of `tmp` is set to the edited string.

```
(define (memorize x)
  (set! tmp x)
  (edit-eval-string "Test Me"
  memorize "Hello how are you?"))
```

`edit-eval-strings` *prompt function string-list* *C* Function

Pops up an edit text box from which the user may edit a list of strings. The text box is labeled with *prompt* and is initialized with the strings in *string-list*, one per line. When the user presses the "Done" button, *function* is called with the edited string list as an argument. The following example prompts the user with a multi-line text editing box labeled `"Hello"`. If the user edits the strings and presses the "Done" button, the value of `tmp` is set to the list of edited strings.

```
(define  (memorize x)
  (set! tmp x)
  (edit-eval-strings "Hello"
  memorize (list "Hi" "Jim "how are you")))
```

`edit-filter-dialog` *old-filter* `ARSdefault.scm`

Edits the filter string in the dialog box and returns the new filter for evaluation in the search-and-replace operation on list of filters.

`eoa` *obj att* — `ARSdefault.scm`

Shorthand function that is synonymous with `(extract-object-attribute)`.

```
>(epoch-time->yyyydoy (say-time-now))
1998084.000000
```

`even?` *num* — `ARSdefault.scm`

Tests if a number is an even integer.

`exec-sql` — *C* Function

Executes an structured query language (SQL) statement and returns `t` if no error occurs.

`exit` — *C* Function

Quits *ARS*. The "Exit" menu item executes this function. The user is prompted to discard or save unsaved editing changes. Objects that are edited but not saved are automatically put on the selection list where the user can easily review them using the "alpha list" menu item. Alternatively, the objects can be saved using the "Save" button on the popup box.

`exit!` — *C* Function

Exits *ARS* immediately, without requesting confirmation.

`expand-window` — `ARSdefault.scm`

Expands the displayed time window by `expand-factor`.

`expand-window-by-factor8` — `IDC.scm`

Expands the displayed time window by a factor of eight so that more detail is shown. The window duration expands symmetrically around the current time window by this factor.

Does not overrides standard *Scheme* function.

`extract-arrival-channel` *arrival* C Function

Extracts and returns the channel object on which the *arrival* occurred. For example:

```
(extract-arrival-channel (find-sole-arrival))
```

`extract-arrivals-channels` *arrival* C Function

Extracts a list of channels that display the specified *arrival*. Arrivals are displayed on channels that are at the same site or array as the detecting channel. The following example returns a list of channels from a common station on which the selected arrival occurred:

```
(extract-arrival-channels
  (find-sole-arrival))
```

`extract-arrival-magauto` *arrival* C Function

Extracts the *magauto* attribute of the designated *arrival*.

`extract-arrival-magdef` *arrival* C Function

Extracts the *magdef* attribute of the designated *arrival*.

`extract-channel-filter-description` *channel* C Function

Extracts the channel filter description of the specified *channel*, expressed as a string of the following five arguments:

1) low breakpoint
2) high breakpoint
3) number of poles
4) filter type
5) causality

Breakpoints are in Hertz. Filter types are defined as follows:

BP – band pass

LP – low pass

HP – high pass

BR – band reject

Causality is defined as follows:

(0) – causal

(1) – acausal

An example return value is `".05 9.0 3 BP 0"`.

`extract-channel-time` *channel* C Function

Extracts the display start time of waveforms from *channel* and returns an epoch time. This time is changed by `zoom-t1-t2` and `pop-zoom-stack`. The following example returns the start time of the waveforms for the selected channel:

```
(extract-channel-time
  (find-sole-arrival))
```

`extract-object-attribute` *object attribute* C Function

Returns the value of an object's attribute given an arrival, channel, origin, or stassoc *object* and a string representing a valid *attribute*. The following example returns the *arid* of the selected arrival:

```
(extract-object-attribute
  (find-sole-arrival) "arid")
```

`extract-object-location` *obj* ARSdefault.scm

Extracts the location of an object, *obj*. If *obj* is a station, the function returns that station's location; if *obj* is an arrival, the function returns the location of the detecting station; if *obj* is an origin, the function returns the origin's location. The location is a list in the form (*latitude longitude depth*).

`filter-channel` *channel* *C* Function

Applies the currently defined filter to the *channel*. (The *ARS* internal variable *filter-parameters* is used to construct the filter.) The following example filters the first channel returned by `(say-channels t)`:

```
(filter-channel
  (car (say-channels t)))
```

`filter-channels-by-distance` *orig chan-list* ARSdefault.scm

Filters the displayed channels based upon whether they are at regional or teleseismic distance from the origin, *orig*.

`filter-hydro-channel` *chan* ARSdefault.scm

Checks for station-specific filters and uses the default if none are found.

`filter-prompt-selected-channels` IDC.scm

Prompts the user to select a filter from a the list-of-filters and cascade-of-filters, then filters selected channels with filter parameters of choice.

The override removes a call to `(sort-filters)`.

`filter-selected-channels` ARSdefault.scm

Maps `filter-channel` to all channels on the selection list.

`find-alpha-list-arrivals` *origin* `IDC.scm`

Examines each origin's associated arrival and determines if the arrival is displayed on the current alpha list, by using the variable `*alpha-list-arrivals*`. This variable is maintained by the function `show-alpha-list-and-remember-arrivals`, which is configured to execute prior to a call to `show-alpha-list`.

This function forces the alpha list functions, which set Azimuth + Slowness non-(`defining set-associated-arrivals-azimuth-defining!`) (`set-associated-arrivals-slowness-defining!`) and Time nondefining (`set-associated-arrivals-time-defining!`), to operate on only arrivals within the current alpha list, rather than all associated arrivals.

Calling objects:
`set-associated-arrivals-azimuth-defining!`
`set-associated-arrivals-slowness-defining!`
`set-associated-arrivals-time-defining!`

Does not override the standard *Scheme* function.

`find-alphalist-object` *object* *C* Function

Returns *object* if it is in the alpha list; otherwise it returns `nil`.

`find-arrivals-interval-channels` *channel-list time duration*
`ARSdefault.scm`

Searches the *channel-list* and returns a list of channels that have arrivals within the time period defined by *time* and *duration*. The returned list is a subset of *channel-list*.

`find-associated-origins` *arrivals* `ARSdefault.scm`

Returns a list of origins that are associated with at least one arrival in the *arrivals* list. Duplicates are not culled.

`find-associated-stassocs` *arrivals* `ARSdefault.scm`

Returns a list of *stassocs* that are associated with at least one arrival in the *arrivals* list. Duplicates are not culled.

`find-create-wftag-waveform` *tagid tagname display-p* — *C* Function

Using the **wftag** table, this function finds the waveform data for *tagid* and *tagname*. *tagid* must be a valid key identifier as specified in the database. *tagname* must be "`arid` ", "`orid`", or "`stassid`". The display variable is `t` or `nil` depending upon whether or not the waveform should be displayed. The following example finds and creates the waveform for *arid* 3345:

```
(find-create-wftag-waveform 3345 "arid" nil)
```

`find-earlier-arrival` *arrival1 arrival2* — `ARSdefault.scm`

Returns the earlier of *arrival1* and *arrival2*.

`find-earliest-arrival` *a-arrival arrival-list* — `ARSdefault.scm`

Returns the earliest arrival from an *arrival-list*. A seed arrival, *a-arrival*, which is usually the car of the *arrival-list*, is also passed to the function.

`find-evid-origin` *evid* — `ARSdefault.scm`

Returns the origin object with the specified *evid*.

`find-history` *str* — *C* Function

Finds a previous command based on the number or matching front characters. *str* must be of the form `!`<*hist*> where <*hist*> is either a number or a string. If it is a number, it must be in the set of known commands; if it is a string, it literally must match the first characters in the commands. This function is not as sophisticated as `CSH!`

`find-if` *cond-func lst* — `ARSdefault.scm`

Finds the first element in the list that satisfies the condition function.

`find-interval-arrivals` *minimum maximum arrival-list* — `ARSdefault.scm`

Searches a list of arrivals and returns those arrivals whose time is within the range defined by *maximum* and *minimum*.

`find-later-arrival` *arrival1 arrival2* `ARSdefault.scm`

Returns the later of *arrival1* and *arrival2.*

`find-latest-arrival` *a-arrival arrival-list* `ARSdefault.scm`

Returns the latest arrival from an *arrival-list*. A seed arrival, *a-arrival*, which is usually the car of the *arrival-list*, is also passed to the function.

`find-origin-associated-arrivals` *origin* *C* Function

Returns a list of all arrivals associated with a specified *origin*. For example:

```
(find-origin-associated-arrivals
  (find-sole-origin))
```

`find-phase-residual-info` *phase delta* `ARSdefault.scm`

Table lookup routine for information in `*defining-phase-residuals-list*`. Returns `t` or `nil` depending if the specified *phase* at distance *delta* satisfies the residual limits in the defined list.

`find-residual-in-table` *the-table assoc* `IDC.scm`

Returns the acceptable list of residuals, given a table of acceptable residual values.

Calling object: `check-assoc-residual`

Does not override the standard *Scheme* function.

`find-scanning-filter-for-channel` *chan* `ARSdefault.scm`

Searchs the variable `*scanning-filters*` for the proper filter parameters. `*scanning-filters*` also contains array and single-station defaults.

`find-sole-arrival` `ARSdefault.scm`

Checks if one and only one arrival exists in the selection list. If so, this function returns the arrival; otherwise, it displays an error message and returns `nil`.

`find-sole-channel` `ARSdefault.scm`

Checks if one and only one channel exists in the selection list. If so, this function returns the channel; otherwise, it displays an error message and displays `nil`.

`find-sole-origin` `ARSdefault.scm`

Checks if one and only one origin exists on the selection list. If so, this function returns the origin; otherwise, it displays an error message and returns `nil`.

`find-sole-stassoc` `ARSdefault.scm`

Checks if one and only one *stassoc* exists on the selection list. If so, this function returns the *stassoc*; otherwise, it displays an error message and returns `nil`.

`find-stassoc-associated-arrivals` *stassoc* *C* Function

Returns the list of arrivals associated with the given *stassoc*.

`find-station-azimuth-reliability` *sta* `ARSdefault.scm`

Lookup routine for azimuth information in `*station-azimuth-slowness-reliability-list*`; if the station is in this list, the function returns `t`; otherwise, it returns `nil`.

`find-station-slowness-reliability` *sta* `ARSdefault.scm`

Lookup routine for slowness information in `*station-azimuth-slowness-reliability-list*`; if the station is in this list, the function returns `t`; otherwise, it returns `nil`.

`find-waveform-inwindow-channels` *channel-list start-time duration*
`ARSdefault.scm`

Searches the *channel-list* and returns a list of channels that have waveforms within the time period defined by *start-time* with the specified *duration*.

`fix-depth` IDC.scm

Sets the cvar `locator-fixed-depth-p` to `"y"` and sets the fixing depth as specified.

Does not override the standard *Scheme* function.

`fix-unzoom-all` IDC.scm

Unzooms all channels, then fixes the time window by unaligning the channels.

Does not override the standard *Scheme* function.

`flatten` *lst* ARSdefault.scm

Condenses a compound nested list of lists into a single list. For example:

```
>(flatten (list (list 1 2) (list 3 4 (list 5)))))
(list 1 2 3 4 5)
```

`float->fixed-string` *float places* *C* Function

Converts a number into a string containing a floating point number, *float*, with *places* number of digits after the decimal point. For example:

```
>(float->fixed-string 123.4567 2)
"123.46"
```

`float->int-string` *float* *C* Function

Converts a floating point number, *float*, to an integer enclosed in a string.

`free-depth` IDC.scm

Complementary function to `(fix-depth)`; it sets the `locator-fixed-depth-p` CVAR to `"n"`.

Does not override the standard *Scheme* function.

`freeze-channels` *flag* — C Function

Inhibits or permits redrawing of waveforms, depending on the value of *flag*. If *flag* is a non-nil value, redrawing is inhibited. For example, `(freeze-channels t)` inhibits redrawing of channels and waveforms and should be followed by a `(freeze-channels nil)` call of a later time.

`frozen?` *object* — C Function

Returns `t` if *object* is frozen (saved); otherwise, it returns `nil`. The following example returns `t` if the selected arrival was saved (written to the output tables):

```
(frozen? (find-sole-arrival))
```

`get-all-associated-channels` *refsta* — `ARSdefault.scm`

Gets all channels associated with a reference station (that is, with the same *refsta*). Returns a list of channel objects.

`get-area-of-interest` *lat lon* — C Function

Given the latitude (*lat*) and longitude (*lon*) of an origin, this function returns a string (not a list containing a string) such as `"AAI"`, `"OSI"`, "SAI", indicating the origin's area of interest. This string is shown below the origin.

`get-assoc-stations` *arrival-objects refstas* — `ARSdefault.scm`

Returns a list of station objects that are associated with the origin from the specified reference stations. An example of a returned list is as follows:

```
((sta1 (chan1 chan2...) delta travel-time prob ) ...)
```

`get-associated-origin` *arrival* — C Function

Returns the origin associated with *arrival*. If *arrival* is unassociated, the function returns `nil`. The following example returns the associated origin for a selected *arrival*:

```
(get-associated-origin (find-sole-arrival))
```

`get-associated-stassoc` *arrival* *C* Function

Returns the *stassoc* associated with *arrival*. If *arrival* is unassociated, the function returns `nil`. The following example returns the associated *stassoc* for a selected arrival:

```
(get-associated-stassoc (find-sole-arrival))
```

`get-beamer-wfdisc-table` *channel-list origin phase-list wfdisc-directory* *C* Function

Prepares *ARS* for on-the-fly beamforming to *origin* using the list of channels in *channel-list*. *origin* is an origin object and *channel-list* is a *LISP* list of channel objects. *phase-list* is a list of phases as strings. The formed beams are written in the *wfdisc-directory*. This function checks the arguments for consistency, sets the busy cursor, and creates the temporary table for *Beamer*.

`get-channel-filter` *description* `ARSdefault.scm`

Checks if a channel filter *description* exists. If so, the description is returned; otherwise, a predefined string representing a non-filter is returned.

`get-channel-info` *channel* `ARSdefault.scm`

Builds a string with *channel* information including station names, time, and filter. An example of a returned string is as follows:

```
"LTX/S2 713433600.000 1 3 2 BP causal"
```

`get-channel-order` *channel* *C* Function

Returns an integer value representing the *channel*'s position within the sorted channel list.

`get-channel-priority` *chan* `IDC.scm`

Returns the sort for a channel as listed in `*channel-ordering*`. The number returned is negative. For example:

```
>(get-channel-priority "sz")
-9.000000
```

Calling object: `compare-sta-chan-distance-priority`

Does not override the standard *Scheme* function.

`get-channels-for-sta-comps` *sta-comps potential-chans* `ARSdefault.scm`

Gets channels for a station and components from a list in the following form: `(`*STA* `(`*comp1 comp2* ...`))`.

`get-channels-max-time` *maximum channels* `ARSdefault.scm`

Returns the maximum channel start time from a list of *channels*. This function is given an initial *maximum* time; if all channels have earlier start times than *maximum*, *maximum* is returned.

`get-channels-min-time` *minimum channels* `ARSdefault.scm`

Returns the minimum channel start time from a list of *channels*. This function is given an initial *minimum* time; if all of the channels have a later start time than the *minimum*, *minimum* is returned.

`get-channels-order` *C* Function

Returns a list of the channels in the order that they are currently displayed.

`get-db-vendor` *C* Function

Returns a string with the name of the database vendor, for example: `"oracle"`.

`get-dbname` *C* Function

Returns the name of the database currently in use.

`get-defining-first-arriving-phases` *associations* `ARSdefault.scm`

Returns a list of the first arriving defining *associations* at each station.

`get-dfx-amp3c-table` *C* Function

Returns a string containing the name of the temporary **amp3c** table used by *Detection and Feature Extraction* (*DFX)*.

`get-dfx-amplitude-table` *C* Function

Returns a string containing the name of the temporary **amplitude** table used by *DFX*.

`get-dfx-apma-table` *C* Function

Returns a string containing the name of the temporary **ampa** table used by *DFX*.

`get-dfx-arrival-table` *C* Function

Returns a string containing the name of the temporary **arrival** table used by *DFX*.

`get-dfx-detection-table` *C* Function

Returns a string containing the name of the temporary **detection** table used by *DFX*.

`get-dfx-hydro-features-table` *C* Function

Returns a string containing the name of the temporary **hydro_features** table used by *DFX*.

`get-filter-override-specs` *sta* `ARSdefault.scm`

Returns the filter override specifications for a given station, *sta*. If no override filter exists for this station, NIL is returned. Filter override values are obtained from the variable `*station-filter-overrides*`. The override structure is as follows:
(*STA regional-filter-spec teleseismic-filter-spec*)

`get-gaim-amplitude-table` *C* Function

Returns a string containing the name of the temporary **amplitude** table used by *GAim*.

`get-gaim-apma-table` *C* Function

Returns a string containing the name of the temporary **ampa** table used by *GAim*.

`get-gaim-arrival-table` *C* Function

Returns a string containing the name of the temporary **arrival** table used by *GAim*.

`get-gaim-in-assoc-table` *C* Function

Returns a string containing the name of the temporary **input assoc** table used by *GAim*.

`get-gaim-in-origerr-table` *C* Function

Returns a string containing the name of the temporary **input origerr** table used by *GAim*.

`get-gaim-in-origin-table` *C* Function

Returns a string containing the name of the temporary **input origin** table used by *GAim*.

`get-gaim-out-assoc-table` *C* Function

Returns a string containing the name of the temporary **output assoc** table used by *GAim*.

`get-gaim-out-origerr-table` *C* Function

Returns a string containing the name of the temporary **output origerr** table used by *GAim*.

`get-gaim-out-origin-table` *C* Function

Returns a string containing the name of the temporary **output origin** table used by *GAim*.

`get-help-string` *function* *C* Function

Returns the help string for the *function* that was added by a prior call to `add-to-help-list`. The *function* is a string containing the function name without arguments, but enclosed in parentheses. The `get-help-string` function returns `nil` if no help string was found. The following example displays the help box for `zoom-t1-t2`:

```
show-string “(zoom-t1-t2)”
  (get-help-string “(zoom-t1-t2)”)
```

`get-hydro-display-channel-string` *sta* `ARSdefault.scm`

Returns the station/channel string for a given station appropriate for a call to `string->channels`. The `*hydro-display-channels*` list is checked. If no entry is found, the input station is concatenated with the `*default-hydro-component*` string.

`get-main-chan-for-sta` *sta* `ARSdefault.scm`

Gets the "network" channel for a particular station. For example:

```
(get-main-chan-for-sta "NRA0")
```

returns an arbitrary channel for "NORES."

`get-max-azimuth-residual-for-phase-dist` *assoc* `ARSdefault.scm`

Returns the maximum azimuth residual for a given phase and distance and is made by performing a table lookup on the structure `*defining-phase-residuals-list*`. If the phase/distance is not found in the structure, or if a residual is NA (-1), a very large number (meaning that any residual value is valid) is returned.

`get-max-slowness-residual-for-phase-dist` *assoc* `ARSdefault.scm`

Returns the maximum slowness residual for a given phase and distance and is made by performing a table lookup on the structure `*defining-phase-residuals-list*`. If the phase/distance is not found in the structure, or if a residual is NA (-1), a very large number (meaning that any residual value is valid) is returned.

`get-max-time-residual-for-phase-dist` *assoc* `ARSdefault.scm`

Returns the maximum time residual for a given phase and distance and is made by performing a table lookup on the structure `*defining-phase-residuals-list*`. If the phase/distance is not found in the structure, or if a residual is NA (-1), a very large number (meaning that any residual value is valid) is returned.

`get-origin-label` *origin* `IDC.scm`

Given an *origin* (not a list containing an origin), this function returns a label string to be shown with the *origin* in the *ARS* window, or `nil` if no label is to be shown. For example:

```
>(get-origin-label (find-sole-origin))
"00:20:51 -24/138"
```

Overrides standard *Scheme* functions. The override provides time information in addition to the standard latitude/longitude.

`get-override-chans` *channels* `ARSdefault.scm`

Given a list of channels, this function returns the list that matches the station and channel listed in the structure `*station-component-overrides*`.

`get-phase-resid-3c-azimuth-residual` *phase-info* `ARSdefault.scm`

Extracts the three-component (3-C) azimuth residual from the *phase-info* list, which has the following components:

(*phase-name min-dist max-dist array-time-residual array-slow-residual array-azimuth-residual 3c-time-residual 3c-slow-residual 3c-azimuth-residual phase-type*)

For example:

```
(list '("Pg" 0.0  4.0  2.0 -1.0 .0  2.0 -1.0 7.5
p-type-primary)
```

`get-phase-resid-3c-slowness-residual` *phase-info* `ARSdefault.scm`

Extracts the three-component (3-C) slowness residual from the *phase-info* list.

`get-phase-resid-3c-time-residual` *phase-info* `ARSdefault.scm`

Extracts the three-component (3-C) time residual from the *phase-info* list.

`get-phase-resid-array-azimuth-residual` *phase-info* `ARSdefault.scm`

Extracts the array azimuth residual from the *phase-info* list.

`get-phase-resid-array-slowness-residual` *phase-info* `ARSdefault.scm`

Extracts the array slowness residual from the *phase-info* list.

`get-phase-resid-array-time-residual` *phase-info* `ARSdefault.scm`

Extracts the array time residual from the *phase-info* list.

`get-phase-resid-max-dist` *phase-info* `ARSdefault.scm`

Extracts the maximum allowable distance from the *phase-info* list.

`get-phase-resid-min-dist` *phase-info* `ARSdefault.scm`

Extracts the minimum allowable distance from the *phase-info* list.

`get-phase-resid-phase` *phase-info* `ARSdefault.scm`

Extracts the phase from the *phase-info* list.

`get-phase-resid-phase-type` *phase-info* `ARSdefault.scm`

Extracts the phase type (for example, p-type-primary) from the *phase-info* list.

`get-phase-type-weights phase-type` *station-type* `ARSdefault.scm`

Looks up the event-confirmation weight records for *phase-type* and *station-type.*

`get-potential-chans` *sta-chan-list* `ARSdefault.scm`

Gets potential channels for display from the master channel list. Computation time is saved in subsequent processing by using this restricted channel list. The function returns a list of channels matching any station and component in the input list. *sta-chan-list* has the following form:

`((`*sta1* `(`*chan1 chan2* `..)) (`*sta2* `(`*chan3 chan4* `..)) ..)`

`get-proper-component-for-station` *sta orig derived-channels*
`ARSdefault.scm`

Returns a list of channels, depending on the station/event distance. *sta* is a list in the following form: `("`*STA*`" (`*chan1 chan2* ...`)` *delta ttime prob*`)`; *orig* is an origin object. The returned channels will be

1) proper channels for distance, or if none exist,

2) a member of the `*default-component-list*`, or if none exist,

3) some component for the station that does exist.

`get-refsta-associated-chans` *refsta* `ARSdefault.scm`

Accessory function that extracts the associated channels from the *refsta* structure (all channels with matching *refsta*).

`get-refsta-chans` *refsta* `ARSdefault.scm`

Accessory function that extracts the channels from the *refsta* structure.

`get-refsta-data-for-origin` *orig refstas* `ARSdefault.scm`

Finds the station distance, travel-time, and detection probability for each station in the input *refstas* list. *orig* is the origin object. *refstas* is a list of stations and channels in the following form:

`((`*sta1* `(`*chan1 chan2* ...`)` `(`*overridechan1 overridechan2*...`)` ...`).`

The function returns a list in the following form:

`((`*sta1* `(`*chan1 chan2*...`)` *overridechan1 overridechan2*...`)`
delta1 ttime1 prob1`)` ...

`get-refsta-delta` *refsta* `ARSdefault.scm`

Accessory function that extracts the delta from the *refsta* structure.

`get-refsta-override-chans` *refsta* `ARSdefault.scm`

Accessory function that extracts the override channels from the *refsta* structure.

`get-refsta-prob` *refsta* `ARSdefault.scm`

Accessory function that extracts the probability of detection from the *refsta* structure.

`get-refsta-refsta` *refsta* `ARSdefault.scm`

Accessory function that extracts the key *refsta* from the *refsta* structure.

`get-refsta-ttime` *refsta* `ARSdefault.scm`

Accessory function that extracts the travel-time from the *refsta* structure.

`get-region-channels` *sta-chan-list* `ARSdefault.scm`

Gets channels for display from station/components in a list of the following form:

`((`*sta1* `(`*chan1 chan2* ..`))` `(`*sta2* `(`*chan1 chan2* ..`))` ..`)`

`get-remarks` *object* — C Function

Returns as a list of strings containing the remarks associated with *object*.

`get-resource-string` *X-resource* — C Function

Retrieves the value of the specified *X-resource* from the X resource database and returns it as a *LISP* string. If *X-resource* is not found, `nil` is returned.

The following example returns the default scaling method:

```
(get-resource-string “*howScale”)
```

`get-special-stations-for-event` *origin refstas* — `ARSdefault.scm`

Generates a list of station *refsta* structures that will be included in the final station set, regardless of the station detection probability. Stations are selected by the following rules:

1) Include all stations in the list **superset-list**; these are stations that analysts always want displayed.
2) Include all stations with distances up to 20 degrees.
3) If the event magnitude is at least 4.5, include all stations with distances up to 120 degrees.
4) If the event magnitude is at least 5.0, include all stations.

`get-sta-channel` *sta* — `IDC.scm`

Returns a channel object whose station matches the specified station name, *sta*. The first channel found is returned; no particular component is guaranteed. For example:

```
>(get-sta-channel "ARA0")
#<ARA0/sz>
```

Calling object: `obj-arrival-compar`

Overrides the standard *Scheme* functions.

`get-station-arrivals` *sta* — `ARSdefault.scm`

Returns a list of all arrival objects for a specific station, *sta*.

`get-station-identifier` *staname* — `ARSdefault.scm`

Finds the station identifier for a given station. If an array element is given, the name of the array is returned, and if a single station name is given, the *refsta* is returned. This function is necessary because the IDC uses arbitrary stations for *refsta*. Examples:

```
>(get-station-identifier "NRA0")
"NORES"
```

```
>(get-station-identifier "MBC")
"MBC"
```

`get-station-identifier-from-object` *obj* — `ARSdefault.scm`

This function is similar to `(get-station-identifier)`, except it takes an *obj*ect for an argument rather than a station.

`get-station-magnitude` *arr orig* — *C* Function

Returns the station magnitude for a specified origin.

`get-string` *object* — `ARSdefault.scm`

Extracts the string representation of the current element. If the element is a number, string, or *ARS* object, it will return the string representation; otherwise, it returns `nil`.

`get-tmp-wfdisc-table` — *C* Function

Returns as a string the name of the temporary **wfdisc** table that *ARS* uses for exchanging waveform data with other programs.

`get-tmp-wftag-table` — *C* Function

Returns as a string the name of the **wftag** table that *ARS* uses for exchanging waveform data with other programs.

`get-weighted-contribution-from-assoc` *assoc* — `ARSdefault.scm`

Returns the sum of the weighted-count contribution from a specific arrival. Only phases belonging to the list `*defining-phase-residuals-list*` will be given weights.

`get-weighted-count` *associations* — `ARSdefault.scm`

Calculates the weighted count, which is used as the Event Definition Criteria for the Reviewed Event Bulletin (REB).

`go` — `IDC.scm`

For the single selected regional origin, sorts the channels according to distance, aligns them on the Pn phase, and zooms the channels so that the associated arrivals are visible.

Does not override the standard *Scheme* function.

`got` — `IDC.scm`

This function is similar to `(go)`, except for it is more oriented to teleseismic events. Alignment is on P rather than Pn, and color-coding is explicitly called.

Does not override the standard *Scheme* function.

`has-remarks?` *object* — *C* Function

Returns `t` if *object* has associated remarks, otherwise, it returns `nil`.

`hide-alpha-list` — *C* Function

Removes the alpha list, but does not destroy its widgets.

`init-blockage-if-necessary` — `ARSdefault.scm`

Creates a function that initializes the blockage files when necessary. "Necessary" means that the input station set contains an uninitialized station, or that the blockage file directory has changed. Station sets and blockage directory are maintained through encapsulation.

`init-refstas` ARSdefault.scm

Returns a list of station names and channels for reference stations. A reference station is a station with *sta* value equal to the *refsta* value. The returned list is of the following form:

((*refsta1* (*chan1* *chan2*...) (*overridechan1* *overridechans2* ...)
(*assoc-chans*) ...)

Channels that are members of `*additional-elements*` or are specified as override channels or associated channels are included. Because arrays have specific names (which can, and usually do, differ from the reference station name), they have to be handled specially. The array names are maintained in the global variable `*array-names*`. Channels will be checked for type *ar* and, if necessary, added to `*array-names*`.

`initialize-wc-station-lists` ARSdefault.scm

Initializes the `*wc-array-list*` list.

`interpret-string` *C* Function

Evaluates the specified string as if it had been submitted to the *Scheme* interpreter. For example:

```
(interpret-string "(+ 3 4)")
7.000000
> t
```

The value `t` is returned after evaluation.

`ipc-create-selectlist-message` ARSdefault.scm

Creates a list of string identifiers for objects on the selection list. This string is used as message data to external programs. Message data consists of the following information:

(*dbname num_origins selected_origins num_stassids elected_stassocs num_detects selected_arrivals num_chans elected_channels channel-times t1 t2*) `is-3-component?` *sta-name* `ARSdefault.scm`

Predicate function that determines if a station is a three-component (3-C) single station.

`is-a-best-channel?` *sta chan* `ARSdefault.scm`

Predicate function that determines if a station (*sta*)/channel (*chan*) is a member of the list `*best-channels*`.

`is-an-additional-element?` *chan* `ARSdefault.scm`

Predicate function that determines if a channel is a member of the list `*additional-elements*`.

`is-an-array?` *sta-name* `ARSdefault.scm`

Predicate function that determines if a station is a member of the list `*wc-array-list*`.

`is-chan-an-array?` *chan* `ARSdefault.scm`

Predicate function that determines if a *chan* is a member of the list `*array-names*`.

`is-hydro-station?` *sta-name* *C* Function

Predicate function that returns `t` if the station name is a hydro-acoustic station.

`is-infra-station?` *sta-name* *C* Function

Predicate function that returns `t` if the station name is an infrasound station.

`is-lang?` *sym* `ARSdefault.scm`

Checks if the language is equivalent to the specified language, *sym*.

`is-shown?` *object* *C* Function

Returns an *object* if that *object* is currently displayed; otherwise, the function returns `nil`.

`is-sta-an-array?` *sta-struct* `ARSdefault.scm`

Predicate function that determines if *sta-struct* is a member of the list `*array-names*`.

`lisp-strcmp` *str1 str2* *C* Function

Compares two strings and returns a number in a manner similar to the *C strcmp* function (see the UNIX manual pages for *strcmp* [3] [IDC6.4Rev1]).

The following example returns `-1` because `“xyzzy”` precedes `“xyzzz”` in alphabetical order.

```
(lisp-strcmp “xyzzy” “xyzzz”)
```

`list->arg-string` *list* `ARSdefault.scm`

Converts a list containing all of one type (a list of numbers, strings or *ARS* objects) to a comma-separated string. For example, if origins 449, 452, and 460 are currently selected,

```
>list->arg-string (say-selected-origins))
“ ‘449’, ‘452’, ‘466’ “
```

`list->comma-string` *l* `ARSdefault.scm`

Converts a list containing objects all of one type (a list of numbers, strings, or *ARS* objects) to a comma-separated string. For example, if origins 217032 and 217033 are on the selectlist,

```
>(list->comma-string (say-selected-origins))
"217032,217033"
```

If the input list is empty, the empty string "" is returned.

`list->string` *number-of-objects-in-list list* *C* Function

Converts a *list* of objects or strings to one string. For example:

```
>(list->string 3 (list (list“4565” “4567”) (list “ ”)
 (list “10260” “10261”)))
“4565 4567 10260 10261”
```

`list-difference` *list-a list-b* `ARSdefault.scm`

Lists all elements of *list-a* that are not in *list-b*.

`list-intersect` *list-a list-b* `ARSdefault.scm`

Returns the list of elements common to two input lists.

`load-init-file` C Function

Searches for and loads the initialization file `ARSdefault.scm`. If the environment variable `$SCHEMEPATH` is set, the variable is used as the search path; otherwise, the following default search path is used:

```
.:/nmrd/dev/scheme/usr/local/lib
```

The function then searches for the file `.ARSinit` in the user's home directory and loads it if found. The function is normally invoked only at start up time. However, it can be used to reload the default definition.

`locate-origin` C Function

Computes a new location for the origin on the selection list by using currently associated arrivals, then stores it in the origin object.

`make-arrival-error-message` *errs* `ARSdefault.scm`

Composes a string comprising the arrival error messages.

`make-origin-error-message` *errs* `ARSdefault.scm`

Composes a string comprising the origin error messages.

`make-string-from-list` *list* `ARSdefault.scm`

Makes a string from *list* elements and ignores members that are not strings or numbers. For example:

```
>(make-string-from-list '("hello" 1 2))
"hello, 1.000, 2.000"
```

`make-string-spaces` *num-spaces* `ARSdefault.scm`

Creates a string *num-spaces* spaces long. For example:

```
>(make-string-spaces 5)
"     "
```

`make-travel-time-path` *path* `ARSdefault.scm`

Strips off the travel-time table basename, leaving only the *path*. For example:

```
>(make-travel-time-path
 data/unclass/ims/pre/rel/ops/data/iasp91/iasp91")
"/data/unclass/ims/pre/rel/ops/data/iasp91/"
```

`make-weighted-count-message` *wt-cnt* ARSdefault.scm

Composes a message confirming or not confirming compliance of an events-weighted count with the threshold established by `*event-confirmation-threshold*`.

`mapcan` *func lyst* `IDC.scm`

Applies the *func*tion to each of the arguments in the specified list and returns a list of the results. For example:

```
> (define (plus3 x) (list (+ x 3)))
#<CLOSURE (x) (list (+ x 3.000000))>
```

```
> (mapcan plus3 (list 1 2 3))
(4.000000 5.000000 6.000000)
```

Calling objects:
`find-alpha-list-arrivals`
`check-origin-for-two-p-phases`

Does not override the standard *Scheme* function.

`max*` *list* `ARSdefault.scm`

Returns the maximum value in the *list*.

`memo` *fn* `ARSdefault.scm`

Creates a "memo-ized" version of an arbitrary function, *fn*, requiring a single argument. "Memoization" is an efficiency technique that stores the function input and output values in a table and uses a table lookup before actually calling the function.

`merge-channels-by-station` *current-chans new-chans* `ARSdefault.scm`

Takes a list of existing channels, finds their displayed order, and merges them into a new set of channels, locating them with the existing station channels.

`modified?` *object* *C* Function

Returns `t` if the *object* was modified; otherwise, returns `nil`. Associating an arrival with an origin causes the origin to be modified and the location hypothesis to be noncurrent. The following example associates a selected arrival with a selected origin and checks if the origin was modified. If `associate-origin-arrival` is successfully completed, it is modified.

```
(associate-origin-arrival
  (find-sole-arrival)
  (find-sole-origin))
(modified? (find-sole-origin))
```

`modify-list-of-remarks` `ARSdefault.scm`

Prompts the user to edit/add to the list of remarks. The response becomes the new list of remarks for the selection box.

`modify-multiple-filters` `ARSdefault.scm`

Prompts the user to edit the entire list of filters in a multi-line edit box. The response becomes the new list of filters.

`modify-object-remarks` *object* `ARSdefault.scm`

Prompts the user to modify the remarks for an object. The response replaces the existing remarks in the object. This pops up a multi-line text entry window with the current remarks in it. If the object is frozen, then no edit is allowed.

`more-menu` *menu maxlen sepint* `ARSdefault.scm`

Automatically creates a "More..." submenu for elements beyond *max-len*. The function limits the number of menu items to *maxlen* and adds a separator for every *sepint* element. Defaults are *maxlen* = `30`, *sepint* = `5`.

`my-extract-arrival-channel` *det* `ARSdefault.scm`

Returns a channel object with the same station name as the input *det*ection. Sometimes `extract-arrival-channel` does not work; this function is slower but should always work.

`my-get-remarks` `IDC.scm`

Returns a list with two elements: the first is a string with a title containing the object type and its identifier, and the second is a string with any associated remarks. For example:

```
> my-get-remarks (find-sole-origin))
("--- Origin 870911" "This is the remark string.")
```

Calling object: `show-remarks-box`

Overrides the standard *Scheme* function, but is local to `show-remarks-box` in `ARSdefault.scm` The override adds station information for arrival remarks.

`new-sort-selected-channels` `IDC.scm`

Moves the selected channels to the top of the display after sorting them by `(compare-sta-chan-distance-priority)`. This function sorts in order of distance, station name, and channel name.

Calling objects:
`qsort-channels-distance-priority`
`qsort-selected-channels-distance-priority`

Does not override the standard *Scheme* function.

`nth` *indx lst* `ARSdefault.scm`

Returns the Nth-indexed element of a list. The list is zero-indexed.

`obj-arrival-compar` *obj1 obj2* `IDC.scm`

Uses the single selected origin to compare two arrival objects. The difference d1 - d2 of the station-event distances is returned. If the stations are at equal distance, the difference in time t1 - t2 is returned. This function is part of the sort routine for the alpha list if `"sort-in-scheme-p"` is set to `True`.

Calling object: `show-alpha-list`

Does not override the standard *Scheme* function.

`obj-sta-compar` *obj1 obj2* `ARSdefault.scm`

String-compares the station attributes of the input two objects; returns the *strcmp*.

`obj-sta-distance-compar` *sta1 sta2* `IDC.scm`

Compares two station objects first by distance, then by alphabetical order. If the first station has priority, a negative number is returned. The function is part of the sort routine for the alpha list if `sort-in-scheme-p` is set to `True`.

Calling objects:
`compare-sta-chan-distance-priority`
`show-alpha-list`

Does not override the standard *Scheme* functions.

`obj-sta-time-compar` *obj1 obj2* `ARSdefault.scm`

String-compares the station attributes of the two input objects; if equal, it compares their times and returns their difference (time1 - time2).

`obj-time-compar` *obj1 obj2* `ARSdefault.scm`

Subtracts the time attributes of the two input objects and returns the difference.

`obj-waveform-in-interval?` *channel time duration* *C* Function

Predicate function that tests if the specified *channel* has data in the time period starting at *time* with *duration* period.

`object->string` *object* *C* Function

Converts an *object* (arrival, channel, origin, or stassoc) to a string. The following example returns a string containing the *orid* of the selected origin:

```
(object->string (find-sole-origin))
```

`origin?` *object* *C* Function

Predicate function that determines whether *object* is an origin. This function returns `t` if the *object* is an origin, or `nil` if it is not. The following example determines if the first object on the selection list is an origin:

```
(origin? (car (say-selectlist)))
```

`*origin-check-functions*` C Function

Collects calls to a set of origin validity-check functions. Currently, it contains only `(check-origin-for-two-p-phases)`.

Calling object: `check-origin-list`

`origin-good-for-hydro-display` *origin* `ARSdefault.scm`

Tests a hydroacoustic origin against the rules that its depth is less than 100, and its seismic region number (SRN) is defined as follows: SRN is less than 7, or SRN is between 8 and 23, or SRN is in (39,43,44,45).

`paint-by-state-object` *object* `ARSdefault.scm`

Paints *object* (origin, stassoc, or arrival) in a manner that differentiates its state.

`paint-code-objects` `ARSdefault.scm`

Applies `(paint-by-state-object)` to all arrivals, displayed channels, origins and *stassocs*.

`paint-color-broken-object` `ARSdefault.scm`

Paints all broken origins.

`paint-color-frozen-object` `ARSdefault.scm`

Applies `(paint-color-frozen)` to all objects (arrivals and origins).

`paint-color-stassoc-unassociated-arrivals` `ARSdefault.scm`

Paints all arrivals that are unassociated with a *stassoc*.

`paint-color-unassociated-arrivals` `ARSdefault.scm`

Paints all arrivals that are unassociated with an origin.

`paint-default-color-object` *object* `ARSdefault.scm`

Paints an *object* the default color.

`paint-default-objects` ARSdefault.scm

Paints all arrivals, displayed channels, origins, and stassocs the default color.

`paint-frozen-object` *object* ARSdefault.scm

Paints all frozen *objects*.

`paint-normal-objects` ARSdefault.scm

Paints all objects, except theoretical arrivals, the default color.

`paint-object` *object color* *C* Function

Changes the foreground color of the *object* to a new *color*. The results are undefined when this function is applied on a monochrome display. The following example paints the selected origin the color purple:

```
(paint-obj (find-sole-origin) "purple")
```

`plot-assoc-hydro-info-on-map` *origin clear-dets blocked-dets*
ARSdefault.scm

Plots an *origin* and associated hydro stations on the Map. Blocked paths and clear paths are drawn as color-coded arcs. The clear paths are drawn elsewhere to handle backpath condition.

`pop-zoom-all` *C* Function

Pops all items on the zoom stack. This function is the "Unzoom All" menu item.

`pop-zoom-stack` *C* Function

Restores the previous time period saved by `push-zoom-stack`.

`populate-dfx-hydro-tables` *C* Function

Populates the **hydro_features** temporary table with the *arid*, modified times, and filter parameters. The temporary table is used to pass data between *ARS* and *DFX* during interactive hydroacoustic recall processing. `t` or `nil` is returned to indicate success or failure. This function does not use arguments.

`populate-dfx-tables` *C* Function

Populates the temporary tables used to pass data between *DFX* and *ARS* for detection recall processing.

`populate-gaim-tables` *C* Function

Populates the temporary tables used to pass data between *GAim* and *ARS* when building events from unassociated arrivals.

`position` *el lst* `ARSdefault.scm`

Returns the position of an element in a list. The first position is 0. If the element is not in the list, this function returns `nil`.

`post-read-database-actions` `ARSdefault.scm`

Performs any work after a database read has completed. This cleanup routine is executed prior to returning control back to the user.

`primary-phase?` *phase* `ARSdefault.scm`

Returns a non-nil value if phase is a primary-phase.

`print-assoc-defs` *assocs* `ARSdefault.scm`

Prints the associations (*sta*tion, *timedef*, *azdef*, and *slodef*) in the input association list, *assocs*.

`prompt-and-rename-arrival` `ARSdefault.scm`

Prompts the user for a new phase name and renames the selected arrival. This is the "Rename" menu item.

`prompt-and-rename-arrivals` ARSdefault.scm

Same as `prompt-and-rename-arrival` except it allows more than one selected arrival.

`prompt-and-select-regional-theoreticals` ARSdefault.scm

Prompts the user with a multi-selection list to select phases to use for the regional theoretical phase display. This function allows mouse-interactive re-definition of the `list-of-regional-theoreticals`, the phases displayed when the "Show Regional Theoretical Arrivals" menu item is selected.

`prompt-and-select-teleseismic-theoreticals` ARSdefault.scm

Prompts the user to select phases from a multi-selection list to use for teleseismic phase display. This function allows mouse-interactive re-definition of the `list-of-teleseismic-theoreticals` phases displayed when the "Show Teleseismic Theoretical Arrivals" menu item is selected.

`prompt-apply-user-choice` *prompt str-list func* *C* Function

Generates a popup selection dialog with an "apply" button in addition to the "done" and "cancel" buttons, so that the dialog will remain up after the selection has been made. For example:

```
(prompt-apply-user-choice "Hello" (list "1" "2" "3") print)
```

The arguments are defined as follows:

prompt	string prompt to appear at the top of the dialog box
str-list	list of selectable strings
func	*Scheme* function to be executed when an item has been selected

`prompt-arrival-fm` ARSdefault.scm

Prompts the user to select from `list-of-fms` to update the *fm* (first-motion) value in the selected arrival.

`prompt-arrival-qual` ARSdefault.scm

Prompts the user to select from `list-of-quals` to update the *qual* value in the selected arrival.

`prompt-arrival-stype` ARSdefault.scm

Prompts the user to select from `list-of-stypes` to update the *stype* value in the selected arrival.

`prompt-button-choice` *prompt str-list func* *C* Function

Generates a popup dialog with radio buttons for the available selections. This function is similar to `prompt-user-choices`, but the selections are presented as rows and columns of radio buttons instead of a long list. The arguments are defined as follows:

prompt	string prompt to appear at the top of the dialog box
str-list	list of selectable strings (must be at least nine elements)
func	*Scheme* function to be executed when an item has been selected

`prompt-button-choices` *prompt str-list* *C* Function

Generates a popup dialog with toggle buttons for the available selections. This function is similar to `prompt-button-choice`, except that more than one item can be selected. The arguments are defined as follows:

prompt	string prompt to appear at the top of the dialog box
str-list	list of selectable strings (must be at least nine elements)
func	*Scheme* function to be executed when an item has been selected

`prompt-channel-theoreticals` IDC.scm

Prompts the user to select a single phase; after selection, the theoretical arrivals for that phase are displayed on the selected channels.

Does not override the standard *Scheme* function.

`prompt-confirmation` *prompt func* — *C* Function

Generates a popup dialog that requests confirmation from the user before executing a *Scheme* function. When calling this function from *Scheme*, `prompt-confirmation` pops up the dialog with confirm and cancel buttons, but control returns immediately to the calling *Scheme* function; it does not wait for the user's selection before returning to the calling *Scheme* function. The calling *Scheme* function should not assume that the requested function has been executed after `prompt-confirmation` returns. The arguments are defined as follows:

prompt	string prompt to appear at the top of the dialog box
func	*Scheme* function to be executed when an item has been selected

`prompt-create-cascade-filter` — `ARSdefault.scm`

Presents the user with a popup box containing a list of filters and cascaded filters. The user can select *N* filters to cascade; these will be added to the list of cascaded filters, but only if the descriptions string is less than 120 characters (see `concat-filter-strings`). Do not combine (`set-cascade-filter`) with (`add-cascade-filter`) because *Scheme* does not wait for *X* input.

`prompt-create-pseudo-origin` — `ARSdefault.scm`

Creates an origin at a location selected by the user. The origin time is estimated from the time of the sole selected arrival.

`prompt-display-station-channels` — `ARSdefault.scm`

Creates a list of all channels associated (through *refsta*) with a selected channel and allows users to select the channels they wish to add to the display. The current *ARS* display channels are not affected by this function.

`prompt-delete-pseudo-origin-location` — `ARSdefault.scm`

Displays the list of origins and allows the user to delete one.

`prompt-display-station-channels` ARSdefault.scm

Creates a list of all channels associated (through *refsta*) with a selected channel and allows users to select all channels they wish to display. Currently displayed channels are not included in the list.

`prompt-filter-choice` *prompt list-of-filters filt-function* *C* Function

Prompts the user to choose a filter from a list in a popup widget; after one is chosen, the filter function is called.

`prompt-filter-dialog` *prompt function string* *C* Function

Prompts the user with a popup box to edit filter parameters. The arguments are defined as follows:

prompt	labels the box
function	function called with the edited string
string	specifies the low breakpoint (number), high breakpoint (number), number of poles (integer), type of filter (string, either "BP", "LP", "HP", or "BR"), and causality of the filter (0 = causal, 1 = acausal).

The following example sets the value of *tmp* to the edited string:

```
(define (memorize x)
  (set! tmp x)
  (prompt-filter-dialog "Test Me"
  memorize "1.2 3.5 4 BP 0"))
```

`prompt-input-locator-depth` *depth-list* *C* Function

Prompts the user for an initial depth to use with the locator. The user is presented with a multiple selection list initialized with the depths (floating-point numbers) in *depth-list*. If *depth-list* is "`nil`", then a default list of depths is used instead. The user also has the option of typing in a specific depth. If the user makes a valid selection, the *Scheme* variable "`locator-depth`" is set.

`prompt-measurement-amptype` ARSdefault.scm

Prompts the user to select from `measurement-amptypes-list` to set the *amptype* for subsequent measurements.

`prompt-object-remarks` `ARSdefault.scm`

Prompts the user to select one or more of the presented remarks to be applied to the selected objects. The strings become remark lines in the database when the object is saved.

`prompt-phase-and-align-channels` *try-designated* `IDC.scm`

Prompts the user for a phase name and aligns all channels on the theoretical arrival time for the returned phase. Checks for only one selected origin and ensures that the origin is valid. This function calls `prompt-phase-name` and passes either the function `align-channels-on-designated-phase` if the channels are to be aligned on the designated phase first or the function `align-channels-on-theoretical-phase` if the channels are to be aligned strictly on the theoretical phase. The list of phases used for the prompt is limited to phases with travel times.

Calling objects:
`prompt-phase-and-align-designated-channels`
`prompt-phase-and-align-theoretical-channels`

Overrides the standard *Scheme* functions; the override uses `"list-of-phases"` instead of `"list-of-phases-with-travel-times"`.

`prompt-phase-and-align-theoretical-channels` `ARSdefault.scm`

Prompts for the phase, then aligns. This is the entry point. The function calls `prompt-phase-and-align-channels` with the value `"nil"` (false) so that the channels will be aligned strictly on the theoretical phases.

`prompt-phase-create-arrival` `IDC.scm`

Prompts the user for a phase name and creates a new arrival at the time t1 with the phase name. This is the "Add Arrival" menu item.

Overrides the standard *Scheme* functions; the override calls `(check-and-create-arrival)` rather than `(prompt-phase-create-arrival)` `prompt-phase-create-idcarrival` ARSdefault.scm

Prompts the user for a phase name and creates a new arrival with the returned phase name at the time t1. This function is a derivative of `prompt-phase-create-arrival`, but with *timedef*, *slodef*, and *azdef* set according to Table 23 on page L-6 of the "International Data Centre" subsection of [GSE95b]. Also, Px, Sx, and tx phases are not associated, even if an origin is selected.

`prompt-phase-name` *function phase-list* ARSdefault.scm

Prompts the user for a seismic phase name from *phase-list* and evaluates the *Scheme function* using the phase name as an argument.

`prompt-phase-names` *function phase-list* ARSdefault.scm

Prompts the user for the selection of zero or more seismic phase names from *phase-list* and evaluates the *Scheme function* using a list of the selected phase names as an argument.

`prompt-phases-send-receive-beamer-messages` IDC.scm

Prompts for one or more phases to be selected from the list of phases and uses the chosen phases and the selected origin and channels to create a message and to send to the *Beamer* program. After receiving a reply, *ARS* displays the generated waveforms as derived channels.

Overrides the standard *Scheme* functions; the override uses `list-of-phases` rather than `list-of-phases-with-travel-times`.

`prompt-read-database` *C* Function

Displays a window that prompts the user for a database name, start time, duration, and network. The user edits the fields and selects "Done" to read the database or "Cancel" to ignore the operation.

`prompt-remark-by-category` IDC.scm

Prompts the user to select a remark category, then to add a remark to be attached to the selected objects.

Does not override the standard *Scheme* function.

`prompt-remark-in-a-category` *remark-category* `IDC.scm`

Prompts the user to select a remark in the specified category. The remark is added to the selected objects. The selected objects and category must agree.

Calling object: `prompt-remark-by-category`

Does not override the standard *Scheme* function.

`prompt-scan-region` `ARSdefault.scm`

Displays the selection box of geographic region choices. The selected region is passed on to the `scan-region` function, which then displays proper channels for that region.

`prompt-set-botf-excluded-chans` `ARSdefault.scm`

Displays a list of all channels associated with the selected channels. All stations selected will be omitted from beam-on-the-fly processing.

`prompt-set-botf-qc-stations` `ARSdefault.scm`

Returns the list of stations that will be quality controlled (QC'd) during beam-on-the-fly processing. Stations not on this list will not be QC'd.

`prompt-show-channels` *C* Function

Prompts the user for channels and displays them. This function is the "Select Channels" menu item.

`prompt-stassoc-etype` `ARSdefault.scm`

Prompts the user to select from `list-of-stassoc-etypes` and updates its *etype* value.

`prompt-user-choice` *prompt list function* C Function

Prompts the user to make a single selection from *list* and calls *function* using the selection as a string argument. For example:

```
(prompt-user-choice
  “Select stassoc etype:”
  list-of-stassoc-etypes
  set-selected-stassoc-etype!)
```

`prompt-user-choices` *prompt list function* C Function

Prompts the user to make multiple selections from a *list* and calls a *function* using the selections as the string argument. For example:

```
(prompt-user-choices
  “Select remarks”
  list-of-remarks
  set-selectlist-remarks!)
```

`purge-dfx-hydro-temp-table` C Function

Removes all entries from the temporary table used for *DFX* hydro features.

`push-zoom-stack` *times* C Function

Saves the displayed time period and zooms to a new period specified by *times* (a two-element list of start and end times relative to the current window). The following example zooms to the time period between t1 and t2, then restores the previous time period:

```
(push-zoom-stack list (say-t1)
  (say-t2) (pop-zoom-stack)
```

`qsort` *compar list* `ARSdefault.scm`

Quicksorts a *list* using a the specified comparison function, *compar*, and returns the sorted list. *compar* is called with two objects to compare; if the first object should precede the second in the sorted list, *compar* should return a number less than zero; if the relative order of the objects does not matter, the function should return zero; if the first object should follow the second, then the *compar* should return a number greater than zero.

`qsort-alpha-channels` `ARSdefault.scm`

Sorts the channel display alphabetically according to station/channel.

`qsort-alpha-list` *chan-compar orig-compar stas-compar arr-compar* `ARSdefault.scm`

Displays an alpha list, sorting the objects according to passed comparison functions. The objects are sorted only if `sort-in-scheme-p` is set to "`true`".

`qsort-channels-distance-priority` `IDC.scm`

Sorts all channels by using the `(compare-sta-chan-distance-priority)` function. This function is used by the `(go)` and `(got)` functions.

Does not override the standard *Scheme* function.

`qsort-channels-zen` `ARSdefault.scm`

Sorts the channels, prioritizing station/sz before station/se before station/sn.

`qsort-distance-channels` `ARSdefault.scm`

Sorts the channel display according to distance from the selected origin.

`qsort-selected-channels` `ARSdefault.scm`

Sorts the channel display, putting the selected channels on top. The sort order within the selected and nonselected sets is preserved.

`qsort-selected-channels-distance-priority` IDC.scm

Synonym for `(new-sort-selected-channels)`.

Does not override the standard *Scheme* function.

`read-database` *name time duration network* *C* Function

Reads the database without displaying the window of read arguments (database *name*, start *time*, *duration*, and *network*). Arguments were previously set up in `prompt-read-database` or configured by the user via *Scheme*.

`read-default-travel-time-tables` ARSdefault.scm

Reads the travel-time tables indicated by the variables `travel-time-tables` and `list-of-phases`.

`read-magnitude-corr-tables` *C* Function

Reads the magnitude correction tables only. Earlier versions of *ARS* combined the reading of travel-time tables and magnitude correction tables.

`read-travel-time-tables-and-locate` IDC.scm

Computes a new location for the selected origin based on associated arrival attributes. Reads the travel-time tables first if necessary.

Overrides the standard *Scheme* functions. The override includes a call to `(check-origin-list)`.

`read-travel-time-tables` *directory-path phase-list* *C* Function

Reads a new set of travel-time tables for the phase in *phase-list* using *directory-path* to find the tables (see `read-default-travel-time-tables`). For example:

```
(read-travel-time-tables
  "/nmrd/top/data/tab"
  list-of-phases)
```

`read-tt-tables-and-align-on-initial-phase` `ARSdefault.scm`

Reads the travel-time tables, if necessary, and aligns on the initial phase. If no initial phases are present, the function aligns on theoretical "P." This function is different from `align-channels-on-phase` because each channel is aligned on one of several "initial" phases instead of all channels being aligned on the same single phase (or theoretical phase if no such phase exists). These phases are defined, in decreasing likelihood of finding, as follows: P, PKPdf, Pn, Pg, and Pdiff.

`recall-channels` `ARSdefault.scm`

Removes all the current channels from the display and displays the nonderived ones previously stored in `stored-channels`. `recall-channels` restores the order of the channels as they were when they were stored.

`recall-selectlist` `ARSdefault.scm`

Clears the current selection list and restores the list that was previously stored with `store-selectlist`.

`receive-message` *address message_id timeout retries* *C* Function

Blocks *ARS* while awaiting the receipt of a specific incoming *message_id*. A dialog box is displayed with the option to abort. For example:

```
(receive-message “Locator” “EndLocator” 0.0 0.0)
```

`receive-XfkDisplay-message` *lyst* `ARSdefault.scm`

Processes a return message from *XfkDisplay*. The *lyst* argument is in the following form:

`( "`*<arid>*`" ("`*<attr-name*`" `*<attr-value>*`) ("`*<attr-name>*`" `*<attr-value>*`)`
`(`*...more name-value pairs....*`) )`

`redraw-ARS` *C* Function

Forces a redraw of the *ARS* display.

`refilter-selected-channels` *filter-parameters* `ARSdefault.scm`

Resets the *filter-parameters* and applies the specified filter to the selected channels.

`refsta-is-hydro-station` *refsta* `ARSdefault.scm`

Returns a non-nil value if an input *refsta* is a hydro station.

`remove-alphalist-object` *object* *C* Function

Removes the specified *object* from the alpha list and its display.

`remove-derived-channels` *make-list-filter* `ARSdefault.scm`

Removes derived channels from a list. The remaining objects are returned in a list.

`remove-if` *cond-func lst* `ARSdefault.scm`

Removes from a list all elements that satisfy the condition function.

`remove-if-not` *cond-func lst* `ARSdefault.scm`

Removes from a list all elements that do not satisfy the condition function.

`remove-selectlist-arrivals-all` `ARSdefault.scm`

Removes all arrivals from the selection list.

`remove-selectlist-channels-all` `ARSdefault.scm`

Removes all channels from the selection list.

`remove-selectlist-object` *object* *C* Function

Removes the arrival, channel, origin, or stassoc *object* from the selection list.

`remove-selectlist-objects-all` *C* Function

Removes all objects from the selection list.

`remove-selectlist-origins-all` *C* Function

Removes all origins from the selection list.

`remove-selectlist-stassocs-all` `ARSdefault.scm`

Removes all stassocs from the selection list.

`rename-and-associate-P-to-origin` `IDC.scm`

Renames the single selected arrival to the phase name specified in the default-phase. Then, if a single origin is selected, associates this arrival with it.

Does not override the standard *Scheme* function.

`rename-and-associate-PKP-to-origin` `IDC.scm`

Renames the single selected arrival to PKP. Then, if a single origin is selected, associates this arrival with it.

Does not override the standard *Scheme* function.

`rename-arrival` *phase* `ARSdefault.scm`

Changes the phase name of the single selected arrival to a new *phase* name.

`rename-arrivals` *phase* `ARSdefault.scm`

Changes the phase name of the selected arrivals to a new *phase* name.

`reset-busy-cursor` *C* Function

Restores the normal cursor to the screen if the busy cursor is displayed.

`reset-frozen-channels` *C* Function

Restores normal drawing updates in case channels have been frozen. This function should be called by error-handling routines in functions that call *freeze-channels*.

`reset-loc` `ARSdefault.scm`

Recomputes the origin location.

`reset-variables` `ARSdefault.scm`

Resets static variables used by several functions. This function is automatically called when performing a "read-database" after discarding the current data set and before reading new data to eliminate any references to the previous data.

`retime-arrival` `ARSdefault.scm`

Changes the time of the single selected arrival to the time indicated by the t1 time marker.

`retime-hydro-arrival` `ARSdefault.scm`

Sets the new times for the selected hydro arrival indicated by the t1 and t2 time markers on the timebar.

`save-and-reset-phase` `IDC.scm`

Resets the default phase to "P" before saving the current event.

Overrides the standard *Scheme* function.

`say-alphalist` *C* Function

Returns the objects on the alpha list in a *Scheme* list.

`say-alphalist-arrivals` *C* Function

Returns a *Scheme* list containing the arrival objects that are on the alpha list.

`say-alphalist-channels` *C* Function

Returns a *Scheme* list containing the channel objects that are on the alpha list.

`say-alphalist-origins` C Function

Returns a *Scheme* list containing the origin objects that are on the alpha list.

`say-alphalist-stassocs` C Function

Returns a *Scheme* list containing the stassoc objects that are on the alpha list.

`say-arrivals` C Function

Returns a list of all arrivals.

`say-base-duration` C Function

Returns the duration of the current waveform window.

`say-base-start` C Function

Returns the start time of the current waveform window.

`say-botf-excluded-chans` `ARSdefault.scm`

Returns the list of channels to be omitted during beam-on-the-fly processing.

`say-botf-qc-stations` `ARSdefault.scm`

Returns the list of stations that will be quality controlled (QC'd) during beam-on-the-fly processing. Stations that are not on this list will not be QC'd.

`say-channels` *flag* C Function

Returns a list of all of the channels. If *flag* is `t`, only displayed channels are returned. The following example lists all channels in the network:

```
(say-channels nil)
```

The following example lists currently displayed channels:

```
(say-channels t)
```

`say-channel-array-name` C Function

Returns the name of the array with which a given channel is associated.

`say-color-by-state-object` *object* C Function

Determines the proper color of an *object* depending upon its state.

`say-current-duration` C Function

Returns the currently displayed duration in seconds.

`say-current-start-time` C Function

Returns the start time of the current waveform window.

`say-group-delay-correction` C Function

Returns `t` if group delay correction is computed and applied for filtered channels. The group delay correction shifts the display of time series to account for group delay in causal filters.

`say-history` C Function

Prints a string listing recent *Scheme* commands; the most recent command appears first. Each command is numbered, along with the date and start and times that it was submitted. Only commands that are directly submitted to the *Scheme* interpreter are listed; these include both menu commands and typed commands. The returned list is a string formatted for easy reading.

`say-history-list` C Function

Returns a list of recent *Scheme* commands as a list of strings.

`say-hydro-channels` C Function

Returns a list of the hydroacoustic channels.

`say-info` *list* `ARSdefault.scm`

Prints an informative string in the popup info box and waits for confirmation. The argument is a list of double-quoted strings.

`say-infra-channels` *C Function*

Returns a list of the infrasonic channels.

`say-max-number-same-selected-arrivals-per-station` `ARSdefault.scm`

Scans the selected arrivals and extracts the station attribute to determine the number of selected arrivals per station. The function returns the maximum number of selected arrivals associated with a station.

`say-number-selected-arrivals` *C Function*

Returns the number of arrivals on the selection list.

`say-number-selected-channels` *C Function*

Returns the number of channels on the selection list.

`say-number-selected-origins` *C Function*

Returns the number of origins on the selection list.

`say-number-selected-stassocs` *C Function*

Returns the number of stassocs on the selection list.

`say-object-remarks` *object* *C Function*

Prints the *object*'s remarks. The function `modify-object-remarks` may provide a better way to view an object's remarks because it allows the user to modify the remarks if necessary.

`say-origin-interval` `ARSdefault.scm`

Calculates and returns the zoom interval for the sole origin. This function is similar to `zoom-on-origin`, but does not actually zoom. This restriction prevents unnecessary zooms when aligning the channels.

`say-origins` *C Function*

Returns a list of all origins.

`say-previous-zoom-duration` *C* Function

Returns the duration of the previous zoom level. At the first zoom level, it returns the current duration. This is useful for `unzoom` because `auto-detail` can anticipate what will be drawn and therefore minimize the amount of graphics needed.

`say-seismic-channels` *C* Function

Returns a list of the seismic channels.

`say-selected-arrivals` *C* Function

Returns a list of all arrivals on the selection list.

`say-selected-channels` *C* Function

Returns a list of all channels on the selection list.

`say-selected-frozen-arrivals` `ARSdefault.scm`

Returns a list of selected, frozen arrivals.

`say-selected-frozen-originss` `ARSdefault.scm`

Returns a list of selected, frozen arrivals.

`say-selected-origins` *C* Function

Returns a list of all origins on the selection list.

`say-selected-stassocs` *C* Function

Returns a list of all stassocs on the selection list.

`say-selectlist` *C* Function

Returns a list of all objects on the selection list.

`say-selectlist-remarks` `ARSdefault.scm`

Prints remarks for every object on the selection list.

`say-selectlist-stats` ARSdefault.scm

Prints in a pop-up dialog box the number of origins, stassocs, channels, and arrivals in the selection list.

`say-sole-arrival` *C* Function

Returns the single selected arrival. If no arrival or more than one arrival is selected, `nil` is returned. This function differs from `(find-sole-arrival)` in that error messages are not printed; only `nil` is returned.

`say-sole-channel` *C* Function

Returns the single selected channel. If no channel or more than one channel is selected, `nil` is returned. This function differs from `(find-sole-channel)` in that error messages are not printed; only `nil` is returned.

`say-sole-origin` *C* Function

Returns the single selected origin. If no origin or more than one origin is selected, `nil` is returned. This function differs from `(find-sole-origin)` in that error messages are not printed; only `nil` is returned.

`say-sole-stassoc` *C* Function

Returns the single selected stassoc. If no stassoc or more than one stassoc is selected, `nil` is returned. This function differs from `(find-sole-stassoc)` in that error messages are not printed; only `nil` is returned.

`say-stassocs` *C* Function

Returns a list of all stassocs.

`say-t1` *C* Function

Returns the time of the t1 marker on the timebar. This time is relative to the start time of the display.

`say-t1-t2` *C* Function

Returns the timebar markers, t1 and t2, as a pair of numbers.

`say-t2` C Function

Returns the time of the t2 marker on the timebar. This time is relative to the start time of the display.

`say-time-until-idle` *time* C Function

Reports the time between *time* and the next time the X event loop is idle. *time* should be an epoch time. This function is used mainly for timing graphic functions. For example:

```
(say-time-until-idle (say-time-now))
```

`say-unassociated-stassoc-phase-arrivals` *phase arrival* `ARSdefault.scm`

Returns a list of *arrivals* that are unassociated with any stassocs with a specified *phase*.

`say-version` `ARSdefault.scm`

Prints the version string of the `ARSdefault.scm` file.

`say-wftag-select-enable` C Function

Returns `t` if orid-selective display of origin beams is enabled. See `(set-wftag-select-enable!)`.

`say-zoom-level` C Function

Returns the current zoom level. Every time `zoom-channels-push` is called, the zoom level increases, and every time `zoom-channels-pop` is called the zoom level decreases. A zoom level of zero indicates that the display is back at the starting interval.

`sbc-sort` C Function

This sort function for `(show-best-channels)` takes the place of the default qsort function, which is slower but more general.

`scale-waveform-height` *factor* — *C* Function

Resizes the height of all waveform widgets by *factor*. For example, if each waveform widget is 60 rasters high, then the following example would make each waveform widget 120 rasters high:

```
(scale-waveform-height 2)
```

`scan-region` *region* — `ARSdefault.scm`

Displays proper channels for the region selected in `prompt-scan-region`.

`scroll-channel-visible` — *C* Function

Takes a single channel as an argument and scrolls the waveforms so that that channel is visible.

`secondary-phase?` *phase* — `ARSdefault.scm`

Returns a non-nil value if phase is not a primary phase.

`select-channels-for-selected-stations` — `ARSdefault.scm`

Gets all available components for the selected channel station name and displays a selection box for the user. After the user has selected the desired components, the available station components are added to the displayed channels.

`select-channels-for-selected-stations0` *selected-component-list* — `ARSdefault.scm`

Gets a list of channels with selected channel identifiers and matching station and reference station names. This function is called after component selection.

`select-coda-and-disassociate` — `IDC.scm`

Adds to the selection list any coda phases (specified in `list-of-coda-phases`) that are associated with the single selected origin and disassociates them.

Does not override the standard *Scheme* function.

`select-filter-dialog` — `ARSdefault.scm`

Selects a filter from the dialog box and calls the edit filter dialog box with this string.

`select-hydro` — `IDC.scm`

Selects all hydro channels.

Does not override the standard *Scheme* function.

`select-infra` — `IDC.scm`

Selects all infra channels.

`select-stations-by-distance` — `ARSdefault.scm`

Sets up a select list from stations ordered by distance and adds selected stations to displayed list. This function uses the `*refstas*` data structure. Only stations with channel data at the expected P arrival time are included.

`select-stations-by-distance0` *stations-distance-list*

`ARSdefault.scm`

This function is called after stations are selected from a pop-up panel. The argument is a list of strings in the following form:

`("`*sta1 refsta1 dist1*`" "`*sta2 refsta2 dist2*`"` ...`)`

`send-and-receive-aeq-messages` — `ARSdefault.scm`

Sends an IPC message to the *AEQ* program and does not wait for a reply.

`send-and-receive-amprev-messages` — `ARSdefault.scm`

Sends an IPC message in libpar format to the *amprev* program, waits for a result, and processes a reply message.

`send-and-receive-arstosac-messages` `ARSdefault.scm`

Sends an IPC message to the *filter* program, waits for a result, and processes a reply message.

`send-and-receive-polariplot-messages` `ARSdefault.scm`

Sends an IPC message to the *Polariplot* program, waits for a result, and processes a reply message.

`send-and-receive-spectraplot-messages` `ARSdefault.scm`

Sends an IPC message to the *Spectraplot* program, waits for a result, and processes a reply message.

`send-geotool-spectogram-message` `IDC.scm`

Sends a message to *Geotool*, instructing it to compute a spectogram for the single selected channel for the time interval designated by the t1 and t2 time markers.

Does not override the standard *Scheme* function.

`send-ipc-message` *address message_id timeout retries* *C* Function

Sends an IPC *message_id* to an *address* containing a message and *retries* with a *timeout* on each try. The *message_id* is known to the receiving process; the *address* is known to the IPC process. The message contains data passed between processes and is usually created with other *Scheme* functions. For example:

```
(send-ipc-message “ARStoSAC” “Begin” 0.0  0.0)
```

`send-MS-orid-message` `IDC.scm`

Initiates the "`RunMsOrid`" message.

`send-Map-messages` `ARSdefault.scm`

Creates a message for the *Map* program; sends par-style messages to the *Map* program for all objects on the selection list. Any stations corresponding to detections are also added to the list.

`send-Map-messages-with-color` `IDC.scm`

Sends to the *Map* program a message to plot all selected objects using the single specified color for all objects.

Does not override the standard *Scheme* function.

`send-receive-DFX-hydro-message` `ARSdefault.scm`

Sends an individual message string created from `get-dfx-hydro-ipc-info` to *DFX*. It first populates the **hydro-features** table with the *arids* and new termination and onset times as well as the filter parameters.

`send-receive-dfx-idc-recall-message` `ARSdefault.scm`

Runs *DFX* to extract the arrival-based features computed during automatic *DFX* processing. This function should be executed after a new arrival has been created. Amplitude, period, and slowness values are computed and may be used in location and magnitude computations.

`send-receive-dfx-recall-message` `ARSdefault.scm`

Creates a new *DFX* message for recall processing by using `(create-dfx-recall-message)`. The function dispatches this message to *DFX,* then receives and processes the results.

`send-receive-gaim-message` `ARSdefault.scm`

Using the sole selected origin as a seed event, this function creates and sends a message to the *GAim* program. *GAim* will search for potential additional detections, which should be associated with this event. The result will be a new event with a new set of associated detections.

`send-selected-hydro-arrivals` *display-flag* `ARSdefault.scm`

Maps the `(display-and-filter-hydro-arrivals)` function to all hydro arrivals on the selection list. The argument determines whether or not to display and filter or undisplay and unfilter.

`send-XfkDisplay-messages` `ARSdefault.scm`

Sends all selected arrivals to *XfkDisplay*.

`send-XfkDisplay-messages-for-beams` `ARSdefault.scm`

Sends all selected arrivals to *XfkDisplay* and creates arrival beams, which are automatically sent back to *ARS*.

`set-amplitude-period-write!` *flag* `ARSdefault.scm`

Enables/disables the ability to measure and set an arrival's amplitude and period. This function is enabled unless *flag* is `nil`. Amplitude and period are measured using the third mouse button: the first click finds the first half of the encompassing rectangle; the second click completes the rectangle. The rectangle's height is 2A; the peak-to-peak amplitude and width are 0.5P or a half-cycle of the signal.

`set-arrival-abs-time!` *arrival time* *C* Function

Sets the time of an arrival to an absolute time. This function is used by the `(retime-arrival)` function.

`set-arrival-bar-off!` *arrival* *C* Function

Removes the detection bar display from *arrival*.

`set-arrival-bar-on!` *arrival* *C* Function

Displays the detection bars from *arrival*.

`set-arrival-magauto!` *arrival magdef* *C* Function

Sets the *magdef* attribute of *arrival* to *magauto*.

`set-arrival-magdef!` *arrival magdef* *C* Function

Sets the *magdef* attribute of *arrival* to *magdef*.

`set-arrival-name!` *arrival phase* *C* Function

Sets the phase name of *arrival* to *phase*.

`set-arrival-time!` *arrival time* *C* Function

Sets the time of *arrival* to *time*. *Time* is relative to the start time of the channel list display.

`set-ars-recovery-path-and-filename` `ARSdefault.scm`

Sets up the path and filename for saving *ARS* recovery data.

`set-associated-arrivals-azimuth-defining!` `IDC.scm`

Sets the arrivals associated with the single selected origin to be azimuth-defining if flag is `t` and not defining if flag is `nil`.

The override updates only arrivals displayed in the current alpha list.

`set-associated-arrivals-slowness-defining!` `IDC.scm`

Sets the arrivals associated with the selected origin to either be slowness defining or slowness nondefining. If its argument is `t`, then *slodef* will be set to defining. If its argument is `nil`, then *slodef* will be set to nondefining.

Override updates only arrivals displayed in the current alpha list.

`set-associated-arrivals-time-defining!` `IDC.scm`

Sets the arrivals associated with the selected origin to either be time defining or time nondefining. If its argument is `t`, then *timedef* will be set to defining. If its argument is `nil`, then *timedef* will be set to nondefining.

Override updates only arrivals displayed in the current alpha list.

`set-channel-howapmeasure!` *channel howapmeasure* *C* Function

Sets the mode for one channel's amplitude/period measurement.

`set-channel-howscale!` *channel howapmeasure* *C* Function

Sets the scaling mode for the selected *channel*.

`set-channel-max-samples!` *channel max* — *C* Function

Sets the maximum number of samples *ARS* will read for the specified channel. If a time window that requires more than *max* samples is displayed, only *max* samples will be plotted, and the remaining time with data will be drawn with a plain horizontal line. This is only referenced the first time a channel is displayed. For example:

```
(set-channel-max-samples! (find-sole-channel) 100000)
```

`set-channel-max-value!` *channel maximum* — *C* Function

Sets the *maximum* plot scale value (in nanometers) for *channel*.

`set-channel-min-value!` — *C* Function

Sets the minimum value (in nanometers) to be used for plotting channels with the waveform widget.

`set-channel-snap!` *channel flag* — *C* Function

Turns on automatic peak location (snapping) for *channel* if *flag* is set to `t`; otherwise it turns it off.

`set-channel-time!` *channel time* — *C* Function

Sets the start *time* of the *channel*'s waveform display trace.

`set-channels-align-all!` *flag* — *C* Function

Specifies whether channel alignment should pertain to only the displayed channels (*flag* = `nil`) or to all channels, displayed or not (*flag* = `t`). The difference is in performance because alignment of undisplayed channels increases the alignment time. However, if all channels are aligned, then displaying new channels will be faster.

`set-channels-default-howscale!` *howscale* — `ARSdefault.scm`

During initial load, this function sets the scaling mode to `auto` for all subsequently created channels. Valid scale-strings are as follows:

`"auto"`	Autoscale when start time, duration, or size changes.
`"resize"`	Autoscale on initialization.
`"fixed"`	Use fixed-scale setting `set-channels-maxscale`.
`"fixedormax"`	Use fixed-scale setting, but auto-rescale if waveforms are too large.

`"auto"` and `"resize"` are most useful; `"fixed"` is similar to `"resize"` except for initialization.

`set-channels-howapmeasure-all!` *howapmeasure* — `ARSdefault.scm`

Sets the mode for amplitude/period measurement. Valid values are `quarter` and `half` for quarter-cycle and half-cycle measurements, respectively. This function sets the mode for all channels that exist at the current time.

`set-channels-howscale` *howscale* — `ARSdefault.scm`

Sets the scaling mode for all channels. Valid *scale-string* are as follows:

`"auto"`	Autoscale when start time, duration, or size changes.
`"resize"`	Autoscale on initialization.
`"fixed"`	Use fixed-scale setting `set-channels-maxscale`.
`"fixedormax"`	Use fixed-scale setting, but auto-rescale if waveforms are too large.

`"auto"` and `"resize"` are most useful; `"fixed"` is similar to `"resize"` except for initialization.

`set-channel-max-samples!` *channel max* — *C* Function

Sets the maximum number of samples that *ARS* will read for the specified channel. If a time window that requires more than *max* samples is displayed, only *max* samples will be plotted, and the remaining time with data will be drawn with a plain horizontal line. This is only referenced the first time a channel is displayed. For example:

```
(set-channel-max-samples! (find-sole-channel) 100000)
```

`set-channels-max-value!` *maxvalue* — `ARSdefault.scm`

Sets the *maximum* value (in nanometers) for the selected channels. This function is mandatory for most uses of fixed scaling, but is ignored for other scaling methods.

`set-channels-order!` *channel-list* — *C* Function

Sets the internal order of known channels to that in *channel-list*. When channels are displayed, they are arranged in internal order.

`set-channels-snap!` *flag* — `ARSdefault.scm`

Enables/disables automatic peak location (snapping) when measuring amplitudes on a waveform. Each waveform is processed and its widget field, which controls snapping, is set. For example:

```
(set-cvar! “snap” “True”)
```
only affects widgets created in the future.

```
set-channels-snap! t
```
sets all channels to “snap.”

```
set-channels-snap! nil
```
sets all channels to “not snap.”

`set-channels-time!` *channels time* — `ARSdefault.scm`

Sets the start time for the channels in the list passed in the argument *channels* to the argument *time*.

`set-color-coding-on!` `ARSdefault.scm`

Displays *ARS* objects with color-coding for easier identification of relationships. This setting is ignored if both the default *ARS* display and the alpha list display do not support color.

`set-color-coding-off!` `ARSdefault.scm`

Does not display the *ARS* objects that have color-coding for easier identification of relationships. This setting is ignored if both the default *ARS* display and the alpha list display do not support color.

`set-default-phase` `IDC.scm`

Displays a list selection box from which the user may select a phase that will be used as the default phase. The variable *default-phase* is set to this phase name.

Does not override the standard.

`set-default-time-def-off` `IDC.scm`

Sets the variable *default-time-def* to "`n`".

Does not override the standard.

`set-default-time-def-on` `IDC.scm`

Sets the variable `default-time-def` to "`d`".

Does not override the standard.

`set-detail-[bar, filter-parameters, phase, scale-type, waveform]-[off!, on!]` `ARSdefault.scm`

Displays detail routines (functions) that adjust the amount of detail displayed for objects. For arrivals, the function shows or does not show vertical time bars, and it shows or does not show phase labels. For waveforms, it shows either a waveform trace or a horizontal line indicating the presence of data. It also shows the type of scaling applied and the filter parameters for the filter applied.

`set-detail-filter-parameters-on` `ARSdefault.scm`

Displays the filter parameters for the filter applied. This function is called directly from within the `IDC.scm` file at load time.

`set-detail-phase-on` `ARSdefault.scm`

Displays phase labels for arrivals. This function is called directly from within the `IDC.scm` file at load time.

`set-detail-hydro-times-off!` `ARSdefault.scm`

Does not display the vertical bracket for onset and termination times.

`set-detail-hydro-times-on!` `ARSdefault.scm`

Displays the vertical bracket for onset and termination times.

`set-detail-object!` *detail-list* *C* Function

Sets the level of detail for graphic objects that can be displayed. *detail-list* is a list of two-item lists, each of which specifies a condition for a graphic object. Valid objects are “`data`”, “`bar`”, “`scale-type`”, and “`filter-parameters`”. Conditions are `t` or `nil`. The following example displays the scale type and waveforms:

```
(set-detail-object!
  (list (list “data” t)
  (list “scale-type” t)))
```

`set-difference` *list1 list2* `ARSdefault.scm`

Returns the set of list elements of *list1* that do not appear in *list2*.

`set-filter-demean!` *chan method* *C* Function

Specifies the demeaning to be applied to waveform data before a filter is applied. *chan* is an *ARS* channel, and *method* is a string that is either `"demean"` or `"linear"`. Any other string will result in no demeaning.

`set-filter-demean-channels-all` `ARSdefault.scm`

Applies the (`set-filter-demean!`) function to all channels.

`set-group-delay-correction!` *C* Function

Specifies whether or not (`t`/`nil`) waveforms should be shifted in time when filtered to account for the filter group delay.

`set-help-mode-on!` *C* Function

Sets the *Scheme* interpreter to the help mode and displays a question-arrow cursor. The next command interpreter is not interpreted, but instead is submitted to the help function, which looks up the help string for the command and displays it. Help mode is automatically disabled after help for one function is displayed.

`set-help-mode-off!` *C* Function

Resets the *Scheme* interpreter from the help mode.

`set-hydro-arrival-time` *arrival* *C* Function

Sets the onset and termination times of the input *arrival* to the times specified by t1 and t2.

`set-hydro-max-samples-10min` `IDC.scm`

Expands the viewing window to 10 minutes, allowing more hydro data to be visible. This function is specific to certain hydro stations.

Does not override the standard *Scheme* function.

`set-hydro-max-samples-16min` `IDC.scm`

Expands the viewing window to 16 minutes, allowing more hydro data to be visible. This function is specific to certain hydro stations.

Does not override the standard *Scheme* function.

`set-hydro-max-samples-default` `IDC.scm`

Returns to the default value of 86000 samples.

`set-non-defining` *assoc type* `IDC.scm`

Updates the list variable `*non-defining-list*`.

Calling objects:

`check-assoc-residual`

`add-residual-error`

Does not override the standard *Scheme* function.

`set-object-attribute!` *object attribute value* *C* Function

Sets the *attribute* of an *object* to *value*. The *attribute* must be a valid field for the *object* and the *value* must be the correct type. Quality assurance (QA) tests are performed on key *attributes*. For example:

```
(set-object-attribute!
  (find-sole-arrival) "arid" 12345)
```

`set-object-remarks!` *object remark-list* *C* Function

Appends the remarks in *remark-list* to the remarks associated with *object*. *remark-list* is a *Scheme* list of strings.

`set-option! name` <*value*> *C* Function

Provides an alternative for setting command-line options. For example:

```
(set-option! "metrics" "/tmp/my_file")
```

This command works as if `-metrics/tmp/my_file` were on the *ARS* command-line. *value* is not necessary if the command-line option does not require an argument.

`set-origin-current!` *origin* *C* Function

Updates the status of origin so that *ARS* considers it "current." Currency requires that all data used to form the *origin* (for example, arrival times and phases) have not been changed since the location was last computed. Only current origins can be saved. This function is useful when external sources have provided the location data. (It typically should not be used.)

`set-phase-amptype-pairs!` *list* — *C* Function

Defines a mapping between phase names and amptypes: The amplitude for **arrival**.*phase* is sought as **arrivalamp**.*amptype*. *list* is a list of pair-strings. For example:

```
(set-phase-amptype-pairs!
  (list "P A5" "PKP A2"))
```

`set-resource-string!` *X-resource value* — *C* Function

Sets the value of the specified *X-resource* to *value*. For example:

```
(set-resource-string!
  "ars*WfdiscWaveform.howScale" "auto")
```

`set-select-arrival-remarks!` *remark-list* — `IDC.scm`

Adds the remarks in *remark-list* to all selected origins.

Calling object: `prompt-remark-in-a-category`

Does not override the standard *Scheme* function.

`set-select-event-remarks!` *remark-list* — `IDC.scm`

Applies remarks to selected origins.

Calling object: `prompt-remark-in-a-category`

Does not override the standard *Scheme* function.

`set-selected-arrival-fm!` *fm* — `ARSdefault.scm`

Sets the first motion attribute of the single selected arrival to *fm*.

`set-selected-arrival-qual!` *qual* — `ARSdefault.scm`

Sets the qual attribute of the single selected arrival to *qual*.

`set-selected-arrival-stype!` *stype* — `ARSdefault.scm`

Sets the stype attribute of the single selected arrival to *stype*.

`set-selected-arrivals-as-defining!` *flag* `IDC.scm`

For selected arrivals, sets *azdef* and *slodef* to be defining if flag is `t` or nondefining if flag is `nil`. *timedef* is not set or unset.

Does not override the standard *Scheme* function.

`set-selected-arrivals-defining!` *flag* `IDC.scm`

For selected arrivals, sets *timedef*, *azdef*, and *slodef* to be defining if flag is `t` or nondefining if flag is `nil`.

Does not override the standard *Scheme* function.

`set-selected-channels-howscale!` *scale-string* `ARSdefault.scm`

Sets the scaling mode for the selected channels. Valid *scale-strings* are defined as follows:

`"auto"`	Autoscale when start time, duration, or size changes.
`"resize"`	Autoscale on initialization.
`"fixed"`	Use fixed-scale setting `set-channels-maxscale`.
`"fixedormax"`	Use fixed-scale setting, but auto-rescale if waveforms are too large.

`"auto"` and `"resize"` are most useful; `"fixed"` is similar to `"resize"` except for initialization.

`set-selected-coda-arrivals-defining!` *flag* `IDC.scm`

For selected phase of Px, Sx, or tx, sets *timedef*, *azdef*, and *slodef* to be defining if flat is `t` or nondefining if flag is `nil`.

Does not override the standard *Scheme* function.

`set-selected-stassoc-etype!` *etype* `ARSdefault.scm`

Sets the etype attribute of the single selected stassoc to *etype*.

`set-selectlist-remarks!` *remark-list* `ARSdefault.scm`

Stores each of the remark strings in *remark-list* with each of the objects on the selection list. The strings are written as remark tuple lines when the object is saved.

`set-selector-select-all!` C Function

Specifies if the next mselector widget used for selecting from a list should start with all items selected by default (`t`). If `nil`, then the mselector will default to just the first item selected.

`set-selector-select-none!` C Function

Specifies if the next mselector widget used for selecting from a list should start with no items selected by default (`t`). If `nil`, then the mselector will default to the first item selected.

`set-t1!` *time* C Function

Sets the t1 timebar marker to *time*; *time* is measured relative to the start time of the display. The following example sets the t1 marker to 11 minutes after the current starting time:

```
(set-t1! 660)
```

`set-t2!` *time* C Function

Sets the t2 timebar marker to *time*; *time* is measured relative to the start time of the display. The following example sets the t2 marker to 20 minutes after the current starting time:

```
(set-t2! 1200)
```

`set-t1-t2!` *t1 t2* `ARSdefault.scm`

Sets the values of the t1 and t2 timebar markers to *t1* and *t2* and moves them to the specified times.

`set-timebar-[on!, off!]` C Function

Specifies whether or not the window's start time is displayed in the timebar window.

`set-timebar-rel-time!` *time* C Function

Sets the relative time of the timebar display to time.

`set-waveform-height!` *size* — *C* Function

Sets the height of each waveform widget to be *size* rasters. The following example sets the size of the waveforms to `100` rasters:

```
(set-waveform-height! 100)
```

`set-wftag-select-enable!` — *C* Function

Turns on (`t`) the origin-selective display of origin beams. This function is called directly from within the `IDC.scm` file during load time. The origin beams are displayed only if the wftag-associated origin is selected. However, if no origins are selected, all origin beams are displayed.

`shift-left-fourth` — `IDC.scm`

Shifts the displayed time interval to be 1/4 of the displayed duration earlier.

Does not override the standard *Scheme* function.

`shift-left-half` — `IDC.scm`

Shifts the displayed time interval to be 1/2 of the displayed duration earlier.

Does not override the standard *Scheme* function.

`shift-right-fourth` — `IDC.scm`

Shifts the displayed time interval to be 1/4 of the displayed duration later.

Does not override the standard *Scheme* function.

`shift-right-half` — `IDC.scm`

Shifts the displayed time interval to be 1/2 of the displayed duration later.

Does not override the standard *Scheme* function.

`shift-window` *shift-fraction* — *C* Function

Shifts the display by *shift-fraction*. A negative value shifts the display left; a positive value shifts it right. The following example shifts the window by 5% (increases the start time by .05):

```
(shift-window .05)
```

The following example shifts the window back 10%:

```
(shift-window -.10)
```

`show-all-cb-channels` — `ARSdefault.scm`

Displays all array cb channels.

`show-alpha-list` — `IDC.scm`

Displays the alpha list with the selected channels, origins, and arrivals.

The override changes *Scheme*-based sorting.

`show-alpha-list-and-remember-arrivals` — `IDC.scm`

Displays the alpha list with the selected channels, origins, and arrivals, then sets the `*alpha-list-arrivals*` variable to the list of selected arrivals.

Calling object: `alpha-assoc`

Does not override the standard *Scheme* function.

`show-and-align-waveform-arrival-channels` *channel-list origin phase1* — `ARSdefault.scm`

For each channel this function a) aligns on phase from origin, b) displays it if it has data, or c) displays it if it has arrivals, else d) does not display it.

`show-and-sense-inwindow-origins` — *C* Function

Shows the *orids* that have arrivals in the displayed window and makes these *orids* selectable. Origins that do not have arrivals in the displayed window are not selectable. This function is used with `show-orid-mask`.

`show-and-sense-inwindow-stassocs` C Function

Shows the *stassocs* that have arrivals in the displayed window and makes these *stassocs* selectable. *Stassocs* that do not have arrivals in the displayed window are not selectable. This function is used with `show-stassoc-mask` (to be defined by the user).

`show-arrival-in-interval-channels` *channels time duration* `ARSdefault.scm`

Finds and displays *channels* that have arrivals within the time period specified by the start *time* and *duration*.

`show-arrival-phase` *arrival* C Function

Shows the phase label for *arrival*.

`show-arrival-with-phase` *arrival phase* `ARSdefault.scm`

Displays the *arrival*'s phase label if the *arrival*'s phase matches *phase*. Returns `t` if any matches occur or `nil` if not.

`show-arrivals-with-phase` *arrivals phase* `ARSdefault.scm`

Displays phase labels for *arrivals* in the list with phase equal to *phase*. Returns `t` if any matches occur or `nil` if not.

`show-bars-origin-associated-arrivals` `ARSdefault.scm`

Shows the arrival bars for all arrivals associated with origins on the selection list. First the bar displays are turned off for all arrivals, then the bar displays are turned on for arrivals associated with each of the origins on the selection list.

`show-bars-stassoc-associated-arrivals` `ARSdefault.scm`

Shows the arrival bars for all arrivals associated with selected *stassocs*. First the bar displays are turned off for all arrivals, then the bar displays are turned on for arrivals associated with each of the *stassocs* on the selection list.

`show-best-chans` ARSdefault.scm

Displays selected channels sorted by distance and aligned by theoretical P-wave arrival. The function uses a selected origin and determines the most effective stations and station component channels to display, based on the probability of detection at each station (for station selection) and event/station distance (for channel selection).

Station components to be displayed are set by the following variables:

`*teleseismic-components-for-display*`

`*regional-components-for-display*`

`*default-component-list*.`

All available members of the teleseismic and regional sets will be displayed. If no channels given by the teleseismic or regional variables are available, a single member of `*default-component-list*` will be shown. If no default member exists, an existing component will be displayed. The displayed channels will be filtered according to the settings of the variables `*teleseismic-filter-parameters*` or `*regional-filter-parameters*`, depending on the station/event distance.

`show-best-hydro-channels` ARSdefault.scm

Checks which hydro stations should show an event and adds the channels to the current display.

`show-busy-cursor` *state* *C* Function

Changes the *ARS* cursor to show a busy state if state is `t` or returns it to its normal state if state is `nil`. For example:

```
(show-busy-cursor! t)

(show-busy-cursor! nil)
```

`show-channel` *channel* *C* Function

Shows the specified *channel*.

`show-chan-theoretical` *phase channel* `IDC.scm`

For the single selected origin, computes and plots the theoretical phase arrival on the specified channel using the standard travel-time tables.

Calling object: `show-selected-channels-theoretical`

Does not override the standard *Scheme* function.

`show-discarded-origin` `IDC.scm`

Displays a window containing the reasons the selected event was discarded.

`show-display-detail` *duration* `ARSdefault.scm`

Turns arrival labels on when time is less than `arrival-on-duration` and off when it is greater. When *duration* is less than `data-on-duration` the waveform display is turned on; otherwise, it is turned off.

`show-help-about-ARS` *C* Function

Displays a popup box with the *ARS* version number and copyright information.

`show-help-about-help` `ARSdefault.scm`

Displays information about using the Help menu.

`show-help-box` *fn_name* `ARSdefault.scm`

Displays a box that shows the string returned by `get-help-string`, with the title of "fn_name".

`show-help-menus` `ARSdefault.scm`

Activates the "Help with menus" mode that causes the first *Scheme* command to be executed, displaying a help box for that command.

`show-help-widget-info` *C* Function

Sets *ARS* into widget selection mode and changes the cursor to a cross. The user may click on any widget in the *ARS* display, and the widget's instance and class names are printed to the *Scheme* window.

`show-help-widgets` `ARSdefault.scm`

Allows the user to select a widget in *ARS* with the mouse to obtain the widget's name and class information. This function is primarily used by developers.

`show-history` *C* Function

Essentially the same function as `(say-history)`, but prints the date and time for both start and end times.

`show-hydro-and-sort-chans` `IDC.scm`

Displays the hydroacoustic channels at top of screen and aligns on t.

Does not override the standard *Scheme* function.

`show-hydro-channels-only` `ARSdefault.scm`

Removes all channels from the display except hydro channels.

`show-hydro-chans` `IDC.scm`

Displays the hydroacoustic channels.

Does not override the standard *Scheme* function.

`show-infra-and-sort-chans` `IDC.scm`

Displays the infrasonic channels at the top of the screen and aligns on I.

`Show-infra-chans` `IDC.scm`

Displays the infrasonic channels.

`show-inwindow-origins` *C* Function

Shows only the origins that have arrivals in the displayed window. This function should be used with `show-orid-mask`.

`show-masked-buttons` `ARSdefault.scm`

Defines masking functions to be applied to the orid and stassid buttons. These masks are typically applied after a zoom operation.

`show-measure-box` *arrival* *C* Function

Displays the amplitude-measurement box for *arrival*.

`show-measurement-box` `ARSdefault.scm`

Displays the amplitude measurement box for the sole selected arrival.

`show-orid-mask` `ARSdefault.scm`

Determines how origins are displayed after functions, such as `zoom-on-origin`, are executed. This function should be defined to point to another function so that the user can make changes.

Let origins be displayed but not selectable:

```
(define (show-orid-mask)
  (show-and-sense-inwindow-origins))
```

Let only those origins with arrivals in the window be displayed:

```
(define (show-orid-mask)
  (show-inwindow-origins))
```

Let all origins be displayed and selectable:

```
(define (show-orid-mask) (no-op))
```

`show-remarks-box` `IDC.scm`

Displays a box with the remarks for the selected objects.

Calling object: `prompt-remark-in-a-category`

Override removes arrival, channel, and origin descriptive information.

`show-selected-channels-theoretical` *phase* `IDC.scm`

Displays the theoretical arrival time for the specified phase on the selected channels.

Calling object: `prompt-channel-theoreticals`

Does not override the standard *Scheme* function.

`show-selected-hbib-channels` `ARSdefault.scm`

Displays the horizontal and incoherent beams (hb and ib channels) for the selected stations if they exist. The channels are re-sorted by distance.

`show-selected-horizontal-channels` `IDC.scm`

Adds to the display short-period or broadband horizontal channels for three-component stations, plus any horizontal beams for array stations. Then, the function sorts the channels by distance.

Does not override the standard *Scheme* function.

`show-selected-objects-waveforms` `ARSdefault.scm`

Creates derived channels and displays the waveform segments for the selected arrival, origin, or stassoc object, using the object's identifier (*arid*, *orid*, or *stassid*) as the tag identifier (*tagid*) index to the **wftag** table.

`show-string` *title string* *C* Function

Displays a popup box with a title and text-box containing the string. The box has a "Done" button, which users can use to remove the box.

`show-string-channels` *string* `ARSdefault.scm`

Removes all channels and shows those from a specified list. This function is similar to `(unshow-and-show-channels)`, except it uses a string argument to designate channels rather than an object list. The string argument has the form `"ARA0/sz WRA/sz GBA/sz"`.

`show-theoreticals` *phases* `ARSdefault.scm`

Deletes all displayed theoretical arrivals, then creates and shows the theoretical arrivals for the phase in the list *phases* for the sole selected origin.

`show-theoreticals-for-regional-phases` `ARSdefault.scm`

Shows the theoretical regional phases for all channels on the current list of theoretical phases.

`show-theoreticals-for-teleseismic-phases` `ARSdefault.scm`

Shows the theoretical teleseismic phases for all channels on the current list of theoretical phases.

`skip-spaces` *string* `ARSdefault.scm`

Returns a string with any leading spaces removed; subsequent embedded spaces are not removed. For example:

```
>(skip-spaces "   1 2")
"1 2"
```

```
>(skip-spaces " 3 5 ")
"3 5 "
```

`slowness->distance` *slowness* *C* Function

Maps a phase-velocity slowness (sec/degree) to an inferred delta (degrees) for an event.

`sort-alpha-channels` *C* Function

Sorts the waveforms alphabetically using a quicksort algorithm.

`sort-by-channel-order` *chan-list chan-order* `ARSdefault.scm`

Sets the channel order as set in input list. *chan-list* is a list of channel objects. *chan-order* is a list of channel strings, for example, (`"sz" "sn" "se"`).

`sort-by-distance-with-channel-order` *orig chan-list chan-order*

`ARSdefault.scm`

Sorts by distance from a given order and sets the channel order as set in the input list. To properly place displayed channels after `show-best-chans` is run (for example, by the use of SlCh), all channels are associated by *refsta*. Arguments are defined as follows:

orig	origin object
chan-list	list of channel objects
chan-order	list of channel strings, for example, ("sz" "sn" "se")

`sort-distance-alpha-channels` C Function

Sorts the alpha list channels first by distance, then by channel name. Sorting by channel name puts cb first, followed by zb and hb, then any other channel ending in 'b'. For copied channels, the "-0n" suffix is ignored.

`sort-distance-channels` C Function

Sorts the channel display according to distance, with the nearest stations first.

`sort-distance-channel-channels` `ARSdefault.scm`

Sorts the channels by distance, then by channel order according to `chan-sort-list`. The default order is as follows:

```
(define chan-list (list "cb" "ib" "zb" "hb" "sz" "sn" "se"))
```

`sort-filters` *filter-list* `ARSdefault.scm`

Sorts a list of filters according to the length of the filter string and the ASCII sort order (or whatever string> uses.)

`sort-obj-list` *obj-list compar-fn* C Function

Sorts the object list according to the comparison function. The comparison function takes two objects as arguments and returns `-1`, `0`, or `+1` depending upon the relative sort order of the two objects.

`sort-selected-channels` *C* Function

Sorts the channel display, putting the selected channels on top. The sort order within the selected and unselected sets is preserved.

`sort-waveform-detect-channels` *C* Function

Uses a quicksort algorithm to sort the waveforms that have data or detections by distance from the single selected origin. Those waveforms that do not have data or detections are ordered last.

`spaces-for-tab` *existing-string tab-position* `ARSdefault.scm`

Returns the number of spaces to create a tab stop at a particular position, given the existing string.

`special?` *object* *C* Function

Returns `t` if the *object* is special; otherwise, it returns `nil`. The special attribute is only valid for channel objects and indicates that the object must remove its *wfdisc* files upon deletion. This attribute is applied to beam channels that have been generated by the user. *object* is a channel object.

`stassoc?` *obj* *C* Function

Tests if the passed object is a stassoc object.

`station-azimuth-reliable?` *sta* `ARSdefault.scm`

Tests whether or not the azimuth measured at the specified station is considered reliable by looking up its value in `*station-azimuth-slowness-reliability-list*`. If the rating is "reliable" then the value (reliable) is returned; otherwise, `nil` is returned.

`station-slowness-reliable?` *sta* `ARSdefault.scm`

Tests whether or not the slowness measured at the specified station is considered reliable by looking up its value in `*station-azimuth-slowness-reliability-list*`. If the rating is "reliable" then the value (reliable) is returned; otherwise, `nil` is returned.

`store-channels` ARSdefault.scm

Stores the current list of nonderived displayed channels for recall later. Derived channels are duplicates of actual channels. This function also stores the order of the channels.

`store-selectlist` ARSdefault.scm

Stores the current list of selected objects for possible recall later.

`str-list->delim-str` *lst delim* ARSdefault.scm

Takes a list of strings and returns a delimited string. For example:

```
> ("FOO" "BAR" "BAZ")
"FOO,BAR,BAZ"
```

`string->arrivals` *arid-str* *C* Function

Given a string containing the *arid* of an arrival, returns the arrival object. If no arrival has that *arid*, `nil` is returned. For example:

```
> (string->arrivals "12864129")
(#<detect 12864129 Pn on WRA/WR_108>)
```

`string->channels` *C* Function

Given a string containing a channel name, this function returns the channel object. If no channel has that *arid*, `nil` is returned. For example:

```
> (string->channels "WRA/fkb")
(#<WRA/fkb>)
```

`string->origins` *C* Function

Given a string containing the *orid* of an origin, returns the origin object. If no origin has that *orid*, `nil` is returned. For example:

```
> (string->origins "871443")
(#<origin 871443>)
```

evids, not *orids*, are usually displayed in the origin list.

`string->stassocs` — *C* Function

Given a string containing the *stassid* of a *stassoc*, returns the *stassoc* object. If no stassoc is in that *stassid*, `nil` is returned. For example:

```
> (string->origins "89773")
(#<stassoc 89773>)
```

`string->string-list` *strings* — `ARSdefault.scm`

Takes a string of the form `"1 2 3 5"` and returns a list of the form `("1" "2" "3" "5")`. Leading and multiple spaces are removed.

`string-nth` *n str* — `ARSdefault.scm`

Returns the *n*th word in the string, *str*. For example:

```
>(string-nth 0 "Hello world")
"Hello"
```

`subseq` *lst start end* — `ARSdefault.scm`

Returns the subset of the list (*lst*) given by *start* and *end* elements. The *start* element (0 is first element) is the beginning of the sublist. The *end* element is the first element after the sublist.

`subset?` *list-a list-b* — `ARSdefault.scm`

Checks if *list-a* is a subset of *list-b*.

`sum` *lst* — `ARSdefault.scm`

Sums the numbers in the specified list (*lst*). For example:

```
>sum (list 1 2 3))
6.000000
```

`teleseismic-distance?` *dist* — `ARSdefault.scm`

Returns a non-nil value if *dist*ance (in degrees) is teleseismic.

`trunc` *num* — `ARSdefault.scm`

Truncates the decimal portion of the *num*ber.

`tt-phase` — `IDC.scm`

Displays a list of phases and allows the user to select one or more; then, shows the theoretical phases for them.

Does not override the standard *Scheme* function.

`unalign-channels-all` — *C* Function

Resets the start time of all channels to the same time.

`undiscard-origin` — `IDC.scm`

Remove the sole selected origin's entry from the **DISCARD** table.

`undo-rename-arrival` — `ARSdefault.scm`

Undoes the last `rename-arrival` function.

`undo-rename-arrivals` — `ARSdefault.scm`

Undoes the last `rename-arrivals` function.

`undo-retime-arrival` — `ARSdefault.scm`

Undoes the last `retime-arrival` function.

`unfilter-channel` *chan-obj* — *C* Function

Removes the filtering from the specified channel-object, *chan-ob.*

`unfilter-selected-channels` — `ARSdefault.scm`

Maps `unfilter-channel` to all channels on the selection list.

`unfilter-channels-all` — *C* Function

Removes the filtering from all channels.

`unfilter-selected-channels` — *C* Function

Removes the filtering from the selected channels.

`unfreeze-and-reset-phase` IDC.scm

Resets the default phase to "P" while unfreezing selected event.

Overrides the standard *Scheme* function.

`unfreeze-selected-arrivals` *C* Function

Unfreezes the currently selected arrivals. Unfrozen arrivals are deleted from the output tables.

`unfreeze-selected-origins` *C* Function

Unfreezes the origins on the selection list. Unfrozen origins are deleted from the output tables.

`unfrozen?` *obj* *C* Function

Predicate function that tests if the specified object, *obj*, is frozen; returns `t` or `nil`.

`unshow-and-show-channels` *lyst* `ARSdefault.scm`

Undisplays all channels and then redisplays only the channels in the specified list.

`unshow-arrival-phase` *arr-obj* *C* Function

Undisplays the phase label of the specified arrival-object, *arr-obj*, from the display.

`unshow-arrival-with-phase` *arrival phase* `ARSdefault.scm`

If the arrival's phase matches the specified *phase,* undisplay the arrival's phase label. This function returns `t` if a match occurred; otherwise, it returns `nil`.

`unshow-arrivals-with-phase` a*rrivals phase* `ARSdefault.scm`

For those arrivals in the list with phase equal to *phase*, this function undisplays their phase labels. It returns `t` if any matches occurred; otherwise, it returns `nil`.

`unshow-channel` *chan-obj* *C* Function

Undisplays the specifed channel-object, *chan-obj*.

`unshow-channeltype` *chantype* `IDC.scm`

Undisplays all channels of a particular type. For example, `(unshow-channeltype "cb")` removes all cb channels from the display.

Calling object: `unshow-horizontals`

Does not override the standard *Scheme* function.

`unshow-channels-all` *C* Function

Undisplays all channels.

`unshow-derived-channels` `ARSdefault.scm`

Undisplays derived channels.

`unshow-horizontals` IDC.scm

Undisplays all horizontal channels.

Does not override the standard *Scheme* function.

`unshow-hydro-chans` IDC.scm

Undisplays the hydroacoustic channels.

Does not override the standard *Scheme* function.

`unshow-infra-chans` IDC.scm

Undisplays the infrasonic channels.

Does not override the standard *Scheme* function.

`unshow-measure-box` *arr-obj* *C* Function

Undisplays the amplitude/period measurement box for the specified arrival. If *arr-obj* is not an arrival, `nil` is returned; otherwise, `t` is returned.

`unshow-measurement-box` `ARSdefault.scm`

Undisplays the amplitude/period measurement box for the single selected arrival.

`unshow-selected-channels` `ARSdefault.scm`

Undisplays the amplitude/period measurement box for the selected arrivals.

`unshow-theoreticals-all` `ARSdefault.scm`

Undisplays all theoretical arrivals.

`unzoom` `ARSdefault.scm`

Unzooms the time scale to the previous setting (the setting before the last zoom operation). `unzoom-all` unzooms successively until the original time window is displayed.

`unzoom-all` `ARSdefault.scm`

Resets the time scale to the original time scale: the start time and duration specified at the database read time.

`user_1arg` *which argument1* *C* Function

General-purpose switch function, which may be `“popen“`, `“stringify,“`, or `“info”`. For example:

```
(user_1arg “popen” “printenv SCHEMEPATH”)
```

`valid-location?` *orig* `ARSdefault.scm`

Predicate that determines if an *orig*in has a non-null location. Null locations are assigned to new origins that have never been located.

`waveform-in-time-interval?l` *chan-obj t1 t2* *C* Function

Returns `t` or `nil` depending on whether *chan-obj* has waveform data within the time interval *t1*-*t2*.

`write-list-database` *obj-list* — *C* Function

Writes all objects on *object-list* to the database.

`write-selectlist-database` — `ARSdefault.scm`

Writes all selected items to the database. This function is typically bound to the "Save" menu item.

`write-window-database` — `ARSdefault.scm`

Writes the following objects displayed in the current window (time period):

1) all origins and associated arrivals in the current window

2) all stassocs and associated arrivals in the current window

3) all unassociated arrivals in the current window

This is the "Save Window" menu item.

`zoom-align-phase-sort-on-origin` *phase* — `ARSdefault.scm`

Composite function that combines:

1) Zoom on Origin.

2) Align on P.

3) Find channels with arrivals in window.

4) Find channels with data in window.

5) Move these channels to the top of the display.

The set of displayed channels is not changed, but is only rearranged according to distance.

`zoom-align-sort-on-origin` — `ARSdefault.scm`

Shortcut for the `zoom-align-phase-sort-on-origin` function; P phase is hardwired.

`zoom-on-arrival` — `ARSdefault.scm`

Zooms to a narrow time period around the sole, selected arrival. The width of the window is specified by `arrival-window-width`.

`zoom-on-origin` — `ARSdefault.scm`

Zooms the display so that all the arrivals of the sole selected origin are visible. The algorithm is as follows:

1) For each arrival that is associated with the origin, compute the offset, which is the difference between the arrival time and the display start time of the channel.
2) Find the minimum and maximum offsets for all arrivals.
3) Use the minimum offset (minus the pre-origin time) as the amount to shift the left of the display.
4) Use the maximum offset (plus the pre-origin time) as the amount to shift the right edge of the display.
5) Compare to minimum duration. If less, then move out end time.

The channel start time to be used to compute the offset is not always obvious. Occasionally, the detecting channel is reported as “_”. Also, when channels are aligned, the start times are adjusted only for the displayed channels. Therefore, if an associated arrival is on a nondisplayed channel and the plot is aligned, the offset algorithm may not work as expected, because the channel start time will be off.

`zoom-on-phase` *period* — *C* Function

Zooms to a given duration around the phase on which the waveforms are aligned. If the waveforms are not aligned, the function performs a regular zoom. *period* is a list containing the start time and end time of the zoom window relative to the start time of the current window.

`zoom-on-stassoc` — `ARSdefault.scm`

Zooms to a narrow time period bounded by the earliest and latest arrival associated with the selected *stassoc*.

`zoom-out` `ARSdefault.scm`

Zooms out to include enough time so that all channels (regardless of alignment shifting) can be seen.

`zoom-select-align-phase-sort-on-origin` *phase* `ARSdefault.scm`

Composite function that combines:

1) Zoom on origin
2) Undisplay all channels
3) For each channel
   a) align on P (find appropriate window times),
   b) display it if it has data, or
   c) display it if it has arrivals, else
   d) do not display it.
4) Sort the channels by distance

This function differs from `(zoom-on-origin-with-noselect)` in that the list of displayed channels is changed when it is invoked.

`zoom-select-align-sort-on-origin` `ARSdefault.scm`

Shortcut function that calls `zoom-select-align-phase-sort-on-origin`, hardwired for the P phase.

`zoom-t1-t2` `ARSdefault.scm`

Modifies the display start time and duration for all channels, incrementing start time by the value in the `t1` timebar marker and setting the duration to the value of t2 - t1.

References

The following sources are referenced in or supplement this document:

[Bow95]	Bowman, R. "Procedures for Seismic Analysis at the GSETT-3 IDC," *1/4/95 memo*, 1995.
[Dzi75]	Dziewonski, A., and Gilbert, F., "The Effect of Small, Aspherical Perturbations on Travel Times and a Re-examination of the Correction for Ellipticity," *Geophys. J. R. astro. Soc.*, 44, 7–17, 1975.
[Eve69]	Evernden, J. F., "Precision of Epicenters Obtained by Small Numbers of Worldwide Stations," *Bull. Seism. Soc. Am.*, Vol 59, pp. 1365–1398, 1969.
[Fli65]	Flinn, E. A., "Confidence Regions and Error Determinations for Seismic Event Location," *Review of Geophysics*, Vol 3, pp 157–185, 1965.
[Fri97]	Friedman, D., and Felleisen, M. *The Little Schemer,* MIT Press, 1997.
[GSE95b]	Group of Scientific Experts, *GSETT 3 Documentation, Volume Two: Operations,* CRP/243, 1995.
[IDC6.2.5]	Science Applications International Corporation, *Analyst Instructions for Seismic, Hydroacoustic, and Infrasonic Data*, SAIC-98/3002, 1998.
[IDC6.4Rev1]	Science Applications International Corporation, *IDC Software Man Pages (Library Functions, User Commands), Revision 1*, SAIC-99/3033, 1999.

[Jor81]	Jordan, T., and Sverdrup, K., "Teleseismic Location Techniques and Their Application to Earthquake Clusters in the South-Central Pacific," *Bull. Seism. Soc. Am.*, 71, 1105–1130, 1981.
[Wan96]	Wang, J., *Analyst Review Station User's Manual*, Science Applications International Corporation, SAIC-96/1100, 1996.

Appendix: Prototype Analysis Tools

This appendix describes the prototype analysis tools and includes the following topics:

- Expand/Unexpand Functions
- Phase Selection Options for Map and AlphaList
- Scanning Aids
- Map <-> ARS Communication Functions

Appendix: Prototype Analysis Tools

This section describes a set of *Scheme* functions that are being considered for addition to `IDC.scm` and `Map.scm` for the *Map* application. The source code is located in `.../config/app_config/interactive/ARS/ARS_proto_tools.scm` and `.../config/app_config/interactive/Map/Map_proto_tools.scm`.

EXPAND/UNEXPAND FUNCTIONS

The following functions give users a detailed look at selected channels or stations.

`(expand-channel)`

Removes all unselected channels from the display and zooms the time scale to t1 and t2.

`(unexpand-channel)`

Restores the channels and time scale that were originally displayed.

`(expand-station)`

Displays all the channels for the station associated with the selected channel. All other channels are removed from the display.

`(unexpand-station)`

Restores the channels that were previously displayed, if the selected channel was an LP channel.

PHASE SELECTION OPTIONS FOR MAP AND ALPHALIST

The following functions add particular categories of phases to the selection list.

`(add-selectlist-unassoc-arrivals-in-window)`

Adds to the selection list all the unassociated arrivals that are visible in *ARS*. The time window that is scanned is from the minimum time of any channel until the latest time of any channel in the window.

`(alphalist-arrivals-all-in-window)`

Clears the selection list, then adds all the arrivals that are visible in the current window. This function scans the time window that is from the minimum time of any channel until the latest time of any channel in the window.

`(alphalist-origin-associated-arrivals)`

Clears the selection list, then adds all arrivals that are associated with the sole selected origin.

`(alpha-inits)`

Clears the selection list, adds the initial arrivals associated with the sole selected origin, then displays the alpha list. This function differs from `(alphalist-origin-associated-arrivals)` in that only initial phases are selected. It is called with the "Alpha P" button.

`(alphalist-unassociated-arrivals-in-window)`

Clears the selection list, adds all the unassociated arrivals that are in the visible window, then displays the alpha list.

SCANNING AIDS

The following functions aid in searching for and creating new events.

`(all-znext)`, `(all-zprev)`

Scrolls the time window and channels to the next (previous) arrival in the data that have been read. The next (previous) arrival is either the one after (before) the sole selected arrival, or if none or more than one is selected, the first arrival after (before) the midpoint of the visible window. The function removes all arrivals from the selection list. This function requires the following *C* functions, which return the parameters of the zoom window lowest on the stack: `(say-base-start)` and `(say-base-duration)`.

`(create-origin-locate)`

Creates a temporary origin, associates all selected arrivals, and computes the surface epicenter. All origins except the new temporary one are removed from the selection list. At least one arrival must be selected. This function sets the following global variables: *cola-arrival*, *start-time*, *time*, *cola-station*, and *cola-period*.

`(delete-blank-waveform)`

Removes a channel if it has no waveform. This internal function is used in `(delete-blank-waveforms)`.

`(delete-blank-waveform-origin)`

Removes a channel if it has no waveform for the sole selected origin. This internal function is used in `(delete-blank-waveforms-event)`.

`(delete-blank-waveforms)`

Removes channels that have no waveform.

`(delete-blank-waveforms-event)`

Removes channels that have no waveform between the origin time and 30 minutes later.

`(display-arrival-channels)`

Displays arrival beams to the sole selected origin for the arrivals on the selection list. If the selected origin has a valid location, the channels are P-aligned.

`(find-next-arrival` *time chan*`)`

Returns the next arrival on station channel, *chan*, after *time*.

`(find-previous-arrival` *time chan*`)`

Similar to `(find-next-arrival` *time chan*`)`, except that this function finds the preceding arrival instead of the next arrival.

`(show-det-beams)`

Removes all channels from the display, then displays the detection beams (fkb channels) for stations in *sp-station-list*. Default filters are automatically applied.

`(uncola)`

Undoes the effect of COLA by

1) restoring the time window,
2) unaligning the channels,
3) removing all origins from the selection list,
4) showing detection beams (fkb channels),
5) setting the waveform height to 45,
6) removing theoretical arrivals, and
7) clearing the selection list.

This function uses the following global variables: *cola-station* and *cola-period*. It does not use *time, start-time*, or *cola arrival*.

`(znext-find-earliest-sta-arrival` *sta lyst*`)`

Returns the earliest arrivals in *lyst* at the specified station. Internal function.

`(znext-find-latest-sta-arrival` *sta lyst*`)`

Returns the latest arrivals in *lyst* at the specified station. Internal function.

`(znext-test-func` *arr*`)`

Same as `(znext-unassoc-test)`. Internal function.

`(znext-unassoc-test` *arr*`)`

Predicate function that is true if an arrival is neither associated nor theoretical. Internal function.

MAP <-> ARS COMMUNICATION FUNCTIONS

These functions facilitate using the *Map* application to select origins or arrivals in the *ARS* application.

The following *ARS* function is directed to *Map*:

`(send-Map-origins)`

Sends all *ARS* origins to the *Map*, which plots them. Existing *Map* objects are retained.

The following *Map* functions are directed to *ARS*:

`(clear-ARS-selectlist)`

Clears the *ARS* selection list.

`(map-days-origins)`

Requests *ARS* to query the database for all origins between 24 hours prior to the current time and the current time.

`(MtA-add-to-selectlist)`

Sends a message to *ARS* requesting that the arrivals, origins, and *origerrs* that are selected in *Map* be added to the *ARS* selection list.

`(MtA-associate-arrivals)`

Clears the *ARS* selection list, sends the *Map*-selected arrivals for addition to the *ARS* selection list, then requests that these arrivals be associated with the sole selected origin in *ARS*. The origin must be previously selected in *ARS*.

`(MtA-clear-and-add-to-selectlist)`

Clears *ARS*'s selection list, then runs `(MtA-add-to-selectlist)` to add the *Map*-selected arrivals, origins, and *origerrs*.

`(MtA-cola)`

Removes arrivals (only) from *ARS*'s selection list, sends the *Map*'s sole selected arrival to *ARS*, then runs `(create-origin-locate-align)` on *ARS*.

`(MtA-cola-and-map)`

Clears *ARS*'s selection list, sends the *Map*'s sole selected arrival to *Map*, runs `(create-origin-locate-align)` on *ARS*, and has *ARS* send all objects on its selection list back to *Map*.

`(MtA-cola-and-map-unassoc)`

Zooms *ARS* to a time window of 60 to 240, runs `(MtA-cola-and-map)`, requests that *ARS* add all unassociated arrivals in the time window to its selection list, then sends all selected objects back to *Map*.

`(MtA-locate)`

Removes origins (only) from *ARS*'s selection list, sends the sole *Map* selected origin to *ARS*, then requests *ARS* to locate that origin.

`(MtA-plot-origins)`

Requests that *ARS* send the *ARS*-selected origins to *Map*.

`(MtA-template-predict)`

Requires that the user has an *ARS*-selected arrival and one *Map*-selected latitude-longitude point. This function will call *ARS*'s `(template-predict)` function, which creates a temporary origin at the *Map* point whose origin time is such that its P arrival is coincident with the *ARS*-selected arrival. In the process, all origins except the new temporary one will be removed from *ARS*'s selection list. In addition, Theoretical P arrivals will be displayed.

`(MtA-zas)`

ARS's selection list is cleared and the *Map*-selected origins are added to it. Then `(zoom-align-sort-on-origin)` is run. This function will fail if not one and only one *Map*-selected origin is on the list.

`(send-selected-arrivals)`

The *arids* of the arrivals on *Map*'s selection list are sent to *ARS*; then *ARS* adds them to its selection list.

`(send-sole-selected-arrival)`

Checks that one and only one *Map* arrival is selected, then runs `(send-selected-arrivals)`.

`(send-selected-origins)`

The *orids* of the origins on *Map*'s selection list are sent to *ARS*; then *ARS* adds them to its selection list.

`(send-sole-selected-origin)`

Checks that one and only one *Map* origin is selected, then runs `(send-selected-origins)`.

`(send-selected-origerrs)`

Calls `(send-selected-origin` *origerrs*`)`.

`(send-selected-originorigerrs)`

Same as `(send-selected-origins)`, except it works on *origerr* objects rather than origin objects. In both cases, *ARS* adds origins to its selection list.

`(send-sole-selected-originorigerr)`

Checks that one and only one *Map origerr* is selected, then runs `(send-selected-originorigerrs)`.

Glossary

A

AEQ

Anomalous Event Qualifier.

Alpha List

(*ARS*) List of stations and phases contributing to a S/H/I event location.

amplitude

Zero-to-peak height of a waveform, in nanometers.

ANSI

American National Standards Institute.

array

Collection of sensors distributed over a finite area (usually in a cross or concentric pattern) and referred to as a single station.

arrival

Signal that has been associated to an event. First, the Global Association (GA) software associates the signal to an event. Later during interactive processing, many arrivals are confirmed and improved by visual inspection.

ARS

Analyst Review Station. This application provides tools for a human analyst to refine and improve the event bulletin by interactive analysis.

ASCII

American Standard Code for Information Interchange. Standard, unformatted 256-character set of letters and numbers.

associate

Assign an arrival to an event.

attribute

(1) Characteristic of an item; specifically, a quantitative measure of a S/H/I arrival such as azimuth, slowness, period, and amplitude. (2) A database column.

azimuth

Direction, in degrees, from a station to an event or seismic signal.

B

back azimuth

Direction, in degrees, from an event or seismic signal to the station.

background noise

Natural movements of the earth as seen on a seismograph, preceding a seismic signal.

beam

Waveform created from array station elements that are sequentially summed in the direction of a specified azimuth and slowness.

Beamer

Application that prepares event beams for the notify process and for later analysis.

branch

Discrete ray paths through the outer and inner core, for example, in the PKPbc branch of the PKP phase.

broken

In ARS, an event is considered broken if one or more of the arrivals associated to it by the automatic processing are associated by the analyst to another event.

build

(1) To create an event by detecting its seismic or hydroacoustic signals, associating its arrivals, identifying them as phases, and locating the event. (2) An operational version of a system or component that incorporates a specified subset of the capabilities that the final product will provide.

bulletin

Chronological listing of event origins spanning an interval of time. Often, the specification of each origin or event is accompanied by the event's arrivals and sometimes with the event's waveforms.

C

°C

Degrees Celsius.

CCB

Configuration Control Board.

cm

Centimeter.

coda phase

Detection found within the envelope of a single phase; an otherwise unidentified phase of unknown path designated as tx, Px, or Sx.

coherent

Having a fixed phase relationship; as signals from a wavefront detected on numerous seismic or infrasonic array station elements.

coherent beam

Summation of data from numerous seismic or infrasonic array station elements after shifting the data traces in time to maximize the coherence of plane-wave signals travelling along a particular azimuth and slowness.

command

Expression that can be input to a computer system to initiate an action or affect the execution of a computer program.

comments

Free text field containing comments made by a station operator or IDC analyst.

component

(1) One dimension of a three-dimensional signal; (2) The vertically or horizontally oriented (north or east) sensor of a station used to measure the dimension; (3) One of the parts of a system; also referred to as a module or unit.

CVAR

A variable stored in a representation that is accessible to both the *C* and *Scheme* segments of a program.

D

DACS

Distributed Application Control System. This software supports inter-application message passing and process management.

defining

Arrival attribute, such as arrival time, azimuth, or slowness, which is used in calculating the event's location or magnitude.

deg.

Degrees of arc (as a distance).

detection

Probable signal that has been automatically detected by the Detection and Feature Extraction (DFX) software.

DFX

Detection and Feature Extraction.

dialog box

Box that appears on the screen after you issue a command and requests information or a decision.

discard

Action of rejecting real or false seismic events that are insufficiently defined according to the PIDC's rules and guidelines.

E

epoch time

Number of seconds after January 1, 1970 00:00:00.0.

event

S/H/I: Unique source of seismic, hydroacoustic, or infrasonic wave energy that is limited in both time and space.

execute

Carry out an instruction, process, or computer program.

exit status

Value returned at the completion of a UNIX command.

F

failure

Inability of a system or component to perform its required functions within specified performance requirements.

false event

Term used to describe events that are not real or have been built by associating noise or nonseismic detections.

F-k

Frequency versus wavenumber (k) analysis that maps phase power from an array as a function of azimuth and slowness.

F-k beam

Coherent beam steered to the azimuth and slowness of the tallest peak in a plot of f-k power.

freeze

To save the results of event processing to the database. This prevents further analysis to the event.

G

GA

Global Association application. GA associates S/H/I phases to events.

GMT

Greenwich Mean Time.

GSE

Group of Scientific Experts.

GSETT

Group of Scientific Experts Technical Test.

GSETT-3

Group of Scientific Experts Third Technical Test.

GUI

Graphical User Interface

H

hb

Horizontal beam.

HF

High frequency.

host

Machine on a network that provides a service or information to other computers. Every networked computer has a hostname by which it is known on the network.

hydroacoustic

Pertaining to sound in the ocean.

Hz

Hertz.

I

ib

Incoherent beam.

ID

Identification; identifier.

IDC

International Data Centre.

IDC Operators	Technical staff that install, operate, and maintain the IDC systems and provide additional technical services to the individual States Parties.
IEEE	Institute for Electrical and Electronic Engineers.
IF	Intermediate frequency.
IMS	International Monitoring System.
IMS Operators	Technical staff that operate and monitor the IMS facilities.
infrasonic	Pertaining to low-frequency (sub-audible) sound in the atmosphere.
interrupt	Keystroke or other signal received by a process that causes it to suspend execution or halt.
IPC	Interprocess communication.
ISC	International Seismic Centre.
ISO	International Standards Organization.

J

Julian date	Increasing count of the number of days since an arbitrary starting date.

K

k	Kilo (prefix), thousand.
KB	Kilobyte. 1,024 bytes.
kg	Kilogram.
km	Kilometer.

L

lat.	Latitude.
LISP	List processing.
LF	Low frequency.
log.	Logarithm.
long.	Longitude.

M

M

Mega (prefix), million.

m

Meter.

Map

Application for displaying S/H/I events, stations, and other information on geographical maps.

MB

Megabyte. 1,024 kilobytes.

m_b

Magnitude of a seismic body wave.

mbmle

Magnitude of an event based on maximum likelihood estimation using seismic body waves.

MF

Medium frequency.

MHz

Megahertz. A million cycles (occurrences, alterations, pulses) per second.

mHz

Millihertz. One millionth of one cycle (occurrences, alterations, pulses) per second.

ML

Magnitude based on waves measured near the source.

mm

Millimeter.

M_s

Magnitude of seismic surface waves.

ms

Millisecond.

Msmle

Magnitude of an event based on maximum likelihood estimation using surface waves.

Mw

Magnitude of an event based on measurements of the moment tensor.

N

N

Nano (prefix), one-billionth.

noise

Incoherent natural or artificial perturbations of the waveform trace.

ns

Nanosecond.

O

onset

First appearance of a seismic or acoustic signal on a waveform.

Operations Manuals

Treaty-specified, formal documents that describe how to provide data, receive IDC products, access the IDC database, and evaluate the performance of the IDC.

Oracle

Vendor of PIDC and IDC database management system.

origin

Hypothesized time and location of a seismic, hydroacoustic, or infrasonic event. Any event may have many origins. Characteristics such as magnitudes and error estimates may be associated with an origin.

P

parameter

Quantitative attribute of a seismic arrival, such as azimuth, slowness, period, and amplitude.

parameter (par) file

ASCII file containing configuration parameters for a program. Par files are used to replace command line arguments. The files is formatted as a list of `[token=value]` strings.

pathname

Filesystem specification for a file's location.

period

Average duration of one cycle of a phase, in seconds per cycle.

phase

Arrival that is identified based on its path through the earth.

PIDC

Prototype International Data Centre.

pipeline

Flow of data at the IDC from the receipt of communications to the final automated processed data before analyst review.

polarity

Direction of first motion on a seismogram; either up (compression) or down (dilatation or relaxation).

polarity reversal

Occurrence of depth-phase waveforms that are mirror images of the initial P-type phases

polarization

Form of three-component analysis used to derive azimuth and slowness information from non-array stations.

pop-up box

Small window that contains selectable objects such as filter settings.

primary seismic

IMS seismic station(s) or data that is (are) part of the detection network.

Q

QC

Quality Control.

R

REB

Reviewed Event Bulletin; the bulletin formed of all events that have passed analyst inspection and quality assurance review.

recovery

Restoration of a system, program, database, or other system resource to a state in which it can perform required functions.

redundant beam

Superimposed waveform trace.

reference channel

Array element to which the station's timing is referenced with respect to its other elements, reflecting the timing of the array as a whole. This channel is typically either the element at the center of a circular array or the element at the intersection of a cross-shaped array.

residual

Difference in time, azimuth, or slowness between a calculated attribute and its corresponding theoretical value.

S

s

Second (time).

SAIC

Science Applications International Corporation.

sampling interval time

Time duration a sample is collected.

SASC

Slowness-Azimuth Station Corrections.

save

Store an analyzed event to the output database, thereby preventing further changes to the event. Same as freezing the event.

scan

Systematically view all waveforms and seek out possible events missed by the automated system.

schema

Database structure description.

Scheme

Interpreted software language by which ARS and other tools are configured.

script

Small executable program, written with UNIX and other related commands, that does not need to be compiled.

select

To choose an element on the screen by clicking on it with the mouse pointer.

SEL1

Standard Event List 1; the bulletin created by total automatic analysis of continuous timeseries data. Typically, the list runs one hour behind real time.

SEL2

Standard Event List 2; the bulletin created by totally automatic analysis of both continuous data and segments of data specifically down-loaded from stations of the auxiliary seismic network. Typically, the list runs five hours behind real time.

SEL3

Standard Event List 3.

S/H/I

Seismic, hydroacoustic, and infrasonic.

SIOD

Scheme In One Defun (an implementation of Scheme).

slowness

Inverse of velocity, in seconds/degree; a large slowness has a low velocity.

SLSD

Standard List of Signal Detections.

snr

Signal-to-noise ratio.

SQL

Structured Query Language; a language for manipulating data in a relational database.

SRID

Sample reference ID.

SRN

Seismic Region Number.

SRST

Source Region Station Time correction (can be used in S/H/I event location).

SSSC

Source specific station correction (can be used in S/H/I event location).

StaPro

Station Processing application for S/H/I data.

station

Site where a monitoring instrument is installed. Stations can either be single sites (for example, BGCA) or arrays (for example, ASAR).

station code (or ID)

(1) Code used to identify distinct stations. (2) Site code.

stepout

Time between two phases, such as pP and P, at a specific station's distance. If the stepout increases as the distance increases, it can be used to identify the phase.

T

theoretical arrival

Point where an arrival is expected to appear on a waveform, based on an event's location and depth.

3-C

Three component as in 3-C station.

timing error

Deviation from absolute time, as measured from a station.

U

unfreeze

Allowing a saved event to be reanalyzed by removing it from the output database.

universal time

Absolute time using Greenwich Mean Time as a reference.

UNIX

Trade name of the operating system used by the Sun workstations.

UTC

Universal Coordinated Time.

V

version

Initial release or re-release of a computer software component.

VMSF

Velocity model specification file (for setting travel time models in S/H/I event location).

Index